MARGINS AND COMBINED CLASSIFIERS

Llew Mason

B. Sc. University of Western Australia

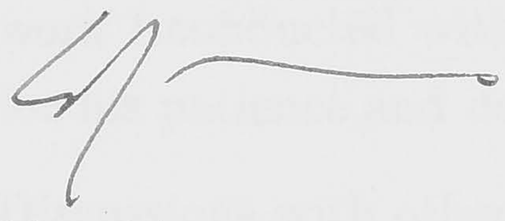
September 5, 1999

A thesis submitted for the degree of Doctor of Philosophy of
The Australian National University

Department of Systems Engineering
Research School of Information Sciences and Engineering
The Australian National University

Declaration

These doctoral studies were conducted with Dr Peter Bartlett and Dr Jonathan Baxter as supervisors, and Professor Robert Bitmead as advisor. The contents of this thesis are the results of original research carried out by myself, in collaboration with others, and have not been submitted for a higher degree at any other university or institution.



Llew Mason

September 5, 1999

Department of Systems Engineering
Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT, 0200
Australia

Acknowledgements

Firstly, I would like to thank my supervisors, Peter Bartlett and Jonathan Baxter, for their professional guidance, enthusiastic support and optimistic outlook throughout the last three years. More importantly, I want to thank them for their friendship, making me feel more like a colleague and less like a student. I also want to thank Mostefa Golea and my advisor, Bob Bitmead, for their input on this thesis.

I gratefully acknowledge the financial support provided by the Australian Postgraduate Award scholarship and the Cooperative Research Centre for Robust and Adaptive Systems.

I want to thank Yoav Freund, Rob Schapire and Yoram Singer for the opportunity to work at AT&T Research Labs during a three month break in my studies. The research work I conducted with Yoav Freund forms part of this thesis and I want to thank him for his patience and dedicated supervision.

Discussions with other researchers, either on their visits to the ANU, or at conferences both in Australia and overseas, have helped to increase my understanding of the issues involved in the work in this thesis. In particular, I want to thank Shai Ben-David, Leo Breiman, Yoav Freund, Jerry Friedman, Gunnar Rätsch, Rob Schapire, Bernhard Schölkopf, Dale Schuurmans, John Shawe-Taylor, Yoram Singer, Alex Smola and Bob Williamson.

The staff and students (both past and present) of Systems Engineering have made life in the department enjoyable. In particular, I want to thank Tanya Conroy, Franky De Bruyne, Subhra Dey, Wayne Dunstan, Jason Ford, Jochen Heinzmann, David Jung, Andrew Lim, Gareth Loy, Jeremy Thorne and Louis Shue. Louis Shue deserves special mention for sharing an office with me during the first two years of my PhD.

The support of my family and friends, especially in light of their confusion as to what I've been doing for the past three years, has been invaluable. I especially want to thank my flat-mates, Rohan Gleeson and Emily Barker, for making our apartment feel like home.

Lastly, I want to thank Savina Kostadinova. Her support and encouragement over the past year has made everything worthwhile. Her efforts at proof-reading much of this thesis are also greatly appreciated.

Abstract

Many binary classification algorithms produce real-valued predictions which are then thresholded to produce a binary classification. These real-valued predictions can often be viewed as a measure of confidence in classification. Recent techniques for the theoretical analysis of such algorithms in terms of the classification confidence (or *margin*) have been dubbed *margins analysis*. This form of analysis has been shown to often yield more refined measures of classifier complexity and more realistic measures of algorithm performance than is provided by the now classical VC theory. This thesis is concerned with the application of margins analysis to classifiers which can be represented as thresholded convex combinations.

Firstly, by representing decision trees as thresholded convex combinations of leaf functions, we apply margins analysis to prove theoretical upper bounds on the generalization error of decision trees and combinations of decision trees in terms of the proportion of training examples with small margin. The measures of classifier complexity suggested by these results can be significantly smaller than those suggested by VC theory. In related work, we describe a new type of classifier, the *alternating decision tree*, which generalizes decision trees, voted decision trees and voted decision stumps. This classifier combines the accuracy of voting methods with the interpretability of decision trees. We present a simple algorithm for constructing alternating decision trees based on boosting and show that this algorithm is competitive with voted decision tree algorithms like C5.0 and yet produces classifiers which are far simpler and easier to interpret.

Secondly, we examine the impressive empirical performance of voting methods (such as the popular AdaBoost algorithm) in terms of margins analysis. We begin by proving a theoretical upper bound on the generalization error of thresholded convex combinations of classifiers involving general cost functions of the margin. We then describe an algorithm, *DOOM*, which adjusts the weights of a voted classifier in order to optimize the cost functions suggested by the theoretical analysis. Experimental results show that this procedure typically improves the generalization error. We then develop an abstract framework for understanding voting methods. Within this framework, many existing algorithms can be viewed as gradient descent optimizers of margin cost functions. For the general class of algorithms suggested by this analysis we provide convergence bounds showing they converge to the optimal cost. By combining this abstract framework with the theoretically motivated cost functions we obtain a new algorithm, *DOOM II*. Experimental results show that DOOM II generally outperforms AdaBoost, particularly in the presence of label noise.

List of publications

Book Chapters

Functional gradient techniques for combining hypotheses
Advances in Large Margin Classifiers, MIT Press, 1999. (to appear)
Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean

Journals

Improved generalization through explicit optimization of margins
Machine Learning, 1999. (to appear)
Llew Mason, Peter Bartlett and Jonathan Baxter

Generalization error of combined classifiers
Journal of Computer and System Sciences, 1999. (to appear)
Llew Mason, Peter Bartlett and Mostefa Golea

Conferences and Workshops

Boosting algorithms as gradient descent
Neural Information Processing Systems 12, 1999. (to appear)
Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean

The alternating decision tree learning algorithm
Proceedings of the Sixteenth International Conference on Machine Learning, 124–133, 1999.
Yoav Freund and Llew Mason

Error bounds for voting classifiers using margin cost functions
IEEE Information Theory Workshop on Detection, Estimation, Classification and Imaging, Santa Fe, 1999.
Llew Mason, Peter Bartlett and Jonathan Baxter

Direct optimization of margins improves generalization in combined classifiers
Advances in Neural Information Processing Systems 11, 288–294, MIT Press, 1999.
Llew Mason, Peter Bartlett and Jonathan Baxter

Generalization in threshold networks, combined decision trees and combined mask perceptrons
Proceedings of the Ninth Australian Conference on Neural Networks, 84–88, 1998.
Llew Mason, Peter Bartlett and Mostefa Golea

Generalization in decision trees and DNF : Does size matter?
Advances in Neural Information Processing Systems 10, 259–265, MIT Press, 1998.
Mostefa Golea, Peter Bartlett, Wee Sun Lee and Llew Mason

Contents

Declaration i

Acknowledgements iii

Abstract v

List of publications vii

List of algorithms xiii

1 Introduction **1**

1.1 Background material 2

1.1.1 Classification model 2

1.1.2 VC theory 4

1.1.3 Voted classifiers 8

1.1.4 Decision tree classifiers 12

1.2 Contributions of this thesis 14

2 Threshold margin bounds for decision trees **17**

2.1 Combinations of classifiers 18

2.2 Decision trees 21

2.3 Optimizing the bound 24

2.4 Experiments and results 26

2.5 Discussion 28

3 Threshold margin bounds for combinations of decision trees **31**

3.1 Combinations of combinations 32
3.2 Combinations of decision trees 34
3.3 Proofs 36
3.3.1 Proof of Theorem 3.1 36
3.3.2 Proof of Theorem 3.2 47
3.4 Discussion 50

4 The alternating decision tree **51**
4.1 Generalizing decision trees 52
4.2 Algorithm 55
4.3 Interpreting alternating decision trees 57
4.4 Experiments and results 59

5 General margin bounds for combined classifiers **65**
5.1 Main result 66
5.2 Cost functions 69
5.3 Discussion 70

6 DOOM - Direct optimization of margins **71**
6.1 Algorithm 72
6.2 Experiments and results 76
6.3 Discussion 78

7 A gradient descent view of voting methods **81**
7.1 Optimizing cost functions of the margin 82
7.1.1 AnyBoost 83
7.1.2 AnyBoost.L_1 85
7.1.3 AnyBoost.L_2 86
7.1.4 AnyBoost and margin cost functionals 87
7.2 Understanding existing voting methods 88
7.3 Convergence results 92
7.3.1 Convergence of AnyBoost 92

7.3.2 Convergence of AnyBoost.L_1 95
7.4 Discussion 97

8 DOOM II **99**
8.1 Algorithm 99
8.2 Experiments and results 103
8.3 Discussion 107

9 Discussion and Conclusions **109**
9.1 Further work 110

Bibliography **113**

List of Algorithms

1.1 AdaBoost [35] 10

4.1 ADTree 56

6.1 DOOM 74

7.1 AnyBoost 84

7.2 AnyBoost.L_1 85

7.3 InnerBoost 87

7.4 MarginBoost 89

7.5 MarginBoost.L_1 90

8.1 DOOM II 102

CHAPTER

1 Introduction

Systems which can learn by example have applications in almost all fields of human endeavour. From financial prediction to face recognition, the practical importance of systems which can adapt to their specific environment cannot be understated. Machine learning is concerned with the study of such systems. This thesis is concerned with perhaps the simplest problem in machine learning, that of classification. The classification problem can be stated as :

> *Given a set of objects which can be separated into several distinct classes and some measurements of these objects, produce a function which, given a new object's measurements, decides to which class the object belongs.*

Some examples of classification problems include :

- Face recognition – An automated security system linked to surveillance cameras. The system learns to recognize personnel by being trained using sample camera images of known personnel. The measurements are an image of a person's face (most likely represented as a two-dimensional array of pixel values) while the classes are 'known personnel' and 'intruder'.

- Fraud detection – An automated customer analysis system used by a company to decide whether customers may be committing fraud. The system is trained using the transaction records of a set of customers known to be either fraudulent or not. The measurements are the transaction records of a customer while the classes are 'fraudulent' and 'not fraudulent'.

- Medical diagnosis – A diagnosis support system which decides, based on patient details (such as height and weight) and medical observations (such as blood pressure), whether a person is at risk

of developing heart disease. The measurements are the patient details and medical observations while the classes are 'high risk' and 'low risk'.

In recent times there has been considerable interest in the field, due largely to the impressive practical performance of techniques such as neural networks, decision trees, support vector machines and voting methods. Understanding the empirical success of these algorithms has been the driving force in theoretical machine learning over the last decade. The tools used in the theoretical analysis of these algorithms have reached a level of maturity that they can now be applied directly to the development of new algorithms which exhibit significantly improved performance. The current state-of-the-art in such analysis is described in several recent books [2, 20, 68, 69, 72]. This thesis is concerned with furthering the development of the theory and practice of these algorithms.

1.1 Background material

1.1.1 Classification model

In order to be able to apply quantitative analysis, we need to formalize the classification problem described above. The classical framework for this problem assumes that examples are independently and identically distributed according to a fixed, *unknown* probability distribution D on $X \times Y$ where X is a measurement space and Y is a label space.

This simple assumption is actually much more restrictive than would appear at first glance. For example, time-series data fails to satisfy this condition since observations are usually dependent. Note however, that this model does *not* assume that there is necessarily some deterministic function $f : X \to Y$ generating class labels. This general model allows for both the measurements and the class labels to be subject to noise. There are many other possible models for classification which we do not consider here. These include models which assume that there exists a target function belonging to a particular class of functions [41, 66], relax the assumption of independently generated examples [1, 7], allow for drift in the generating distribution [4, 8, 33] and relax the assumption that there is a fixed relationship between measurements and class labels [6, 12, 39].

Typically, the measurement space X is taken to be some subset of $\mathbb{R}^n$. For the sake of simplicity, we will almost exclusively deal with the problem of binary classification with $Y = \{\pm 1\}$. The analysis is often significantly more elegant for the binary case than for the multi-class case (where $|Y| > 2$) and most results obtained in the binary case can be easily extended to handle more than two labels. As we shall see shortly, the use of -1 and 1 as class labels makes for simpler notation.

In general, a classification learning algorithm is provided with a sequence S of m labelled

examples $\big((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\big)$. This sequence is generally called a *training set*. Since examples are assumed to be distributed according to D, the training set S is assumed to be distributed according to the product probability distribution D^m. The learning algorithm uses the information contained in S to produce a function $f : X \to Y$ which can hopefully be used to classify new (as yet unseen) examples given only their measurements. For this reason, the function f is known as a *classifier*. Formally, we want to find an f such that $\mathbf{P}_D(f(x) \neq y)$ is small. We use $P_D(A)$ here to denote the probability of choosing according to D an example (x, y) which satisfies a predicate A. That is, we want to choose f such that the probability of a new example being generated (according to D) which cannot be correctly classified by f is small. This quantity is known as the *generalization error* of f.

Ideally we want to find what is known as the *Bayes classifier* f^* defined by

$$f^* = \operatorname*{argmin}_{f:X\to Y} \mathbf{P}_D(f(x) \neq y).$$

The problem of finding f^* is known as *Bayes' problem* and the optimal error, $\mathbf{P}_D(f^*(x) \neq y)$, is called the *Bayes error*. However, since D is unknown, it is necessary to infer the properties of D from the training set S. In practice, this is often accomplished by finding a classifier f from a fixed class of classifiers $\mathcal{F}$ such that $\mathbf{P}_S(f(x) \neq y)$ is small. We use $P_S(A)$ here to denote the proportion of S which satisfies a predicate A. This quantity is known as the *training error* of f.

The question is then, what guarantees can we make about a classifier's generalization error given that we are trying to minimize its training error?

As an example, consider two classifiers f_A and f_B. Classifier f_A is chosen from the class of all possible classifiers $f : X \to Y$. Classifier f_B is chosen from the class consisting of the two constant classifiers which simply return either $+1$ or -1 for all examples. For a particular training set, both f_A and f_B are chosen to minimize training error. Now suppose that the finite training set has zero measure under D and that $\mathbf{P}_D(y = 1) = \mathbf{P}_D(y = -1)$. Clearly, for any training set, f_A can be chosen so that its training error is zero since the class of all possible classifiers $f : X \to Y$ includes the classifier which maps each training example to its corresponding class label. However, the generalization error of this classifier will be 0.5. Classifier f_B's generalization error will also be 0.5, but its training error will converge to 0.5 as the size of the training set increases. The difference in the relationship between training and generalization error for these two classifiers is due to a difference in the *complexity* of the classes from which they are chosen. Since classifier f_A essentially 'memorizes' the training set, the complexity of f_A increases with the size of the training set, while the complexity of classifier f_B is constant.

Intuitively, for any classifier chosen from a class which is not 'too complex', we expect the training error to be a close approximation of the generalization error, provided that

the training set is sufficiently large. To formalize this intuition, we firstly need to define precisely what we mean by the *complexity* of a class of classifiers.

1.1.2 VC theory

In the first part of this section we give some technical results which relate to the complexity of general classes of functions. Later, we will see how these results can be applied to quantify the complexity of classes of classifiers.

Definition 1.1 (Growth Function)
The growth function $\Pi_\Psi(m)$ *of a class* Ψ *of functions* $\psi : X \times Y \to \mathbb{R}$ *is defined by*

$$\Pi_\Psi(m) = \max\left\{|\Psi_{|S}| : S \subseteq X \times Y, |S| = m\right\},$$

where $\Psi_{|S} = \left\{\left(\psi(x_1, y_1), \ldots, \psi(x_m, y_m)\right) : \psi \in \Psi\right\}$ *for* $S = ((x_1, y_1), \ldots, (x_m, y_m))$.

We note that this definition of the growth function is slightly unusual in that we allow functions in Ψ to be real-valued. When the functions in Ψ are restricted to be binary-valued we obtain the usual definition of the growth function as a special case.

We now state a powerful result due to Vapnik and Chervonenkis [70] in a slightly more general form. This result shows that if $\Pi_\Psi(m)$ grows sub-exponentially, then with high probability over choice of S according to D^m, the probability that the expected value of any function in Ψ is much larger than its sample average is small. We use $P_{D^m}(A)$ here to denote the probability of choosing a sequence S of m examples according to D^m which satisfies a predicate A.

Lemma 1.1
For any class Ψ *of functions* $\psi : X \times Y \to [-B, B]$

$$\mathbf{P}_{D^m}\left(\exists \psi \in \Psi : \mathbf{E}_D \psi(x, y) - \mathbf{E}_S \psi(x, y) > \epsilon\right) \leq 4\Pi_\Psi(2m) e^{-m\epsilon^2/2B^2}.$$

The proof of this result essentially mirrors existing proofs for the case where the functions in Ψ are binary-valued. For example, the proof of Theorem 4.3 in [2] can be trivially extended to handle functions in Ψ with bounded range. The only real difference lies in the application of Hoeffding's inequality where the range $\{-1, 1\}$ is replaced by $[-B, B]$, replacing the constant 8 with $2B^2$ in the final result. Since we will make frequent use of Hoeffding's inequality we provide it below for reference.

Lemma 1.2 (Hoeffding's Inequality [40])
Let $X_1, \ldots, X_N$ *be independent bounded random variables such that* X_i *falls in the interval* $[a_i, b_i]$ *with probability one. Let* $S_N = \sum_{i=1}^N X_i$. *Then for any* $\epsilon > 0$ *we have*

that

$$\mathbf{P}\left(S_N - \mathbf{E}(S_N) \geq \epsilon\right) \leq e^{-2\epsilon^2 / \sum_{i=1}^{N}(b_i - a_i)^2}$$

and

$$\mathbf{P}\left(\mathbf{E}(S_N) - S_N \geq \epsilon\right) \leq e^{-2\epsilon^2 / \sum_{i=1}^{N}(b_i - a_i)^2}.$$

Returning to Vapnik and Chervonenkis' result, for the specific case in which functions in Ψ are binary-valued the following result can also be obtained. This result gives an upper bound with a faster rate of convergence on the probability that the expected value of any function in Ψ is much larger than twice its sample average.

Lemma 1.3
For any class Ψ of functions $\psi : X \times Y \to \{0, 1\}$

$$\mathbf{P}_{D^m}\left(\exists \psi \in \Psi : \mathbf{E}_D \psi(x, y) - 2\mathbf{E}_S \psi(x, y) > \epsilon\right) \leq 4\Pi_\Psi(2m)e^{-m\epsilon/8}.$$

Proof

The result follows from the following result (Theorem 2.1 in [3]) due to Vapnik and Chervonenkis [71]. For any class Ψ of functions $\psi : X \times Y \to \{0, 1\}$

$$\mathbf{P}_{D^m}\left(\exists \psi \in \Psi : \frac{\mathbf{E}_D \psi(x, y) - \mathbf{E}_S \psi(x, y)}{\sqrt{\mathbf{E}_D \psi(x, y)}} > \epsilon\right) \leq 4\Pi_\Psi(2m)e^{-m\epsilon^2/4}.$$

Following, for example, the proof of Corollary 7 (iii) in [5], suppose that $\mathbf{E}_D \psi(x, y) - \mathbf{E}_S \psi(x, y) \leq \epsilon\sqrt{\mathbf{E}_D \psi(x, y)}$ and consider the separate cases for which $\mathbf{E}_D \psi(x, y) < 4\epsilon^2$ and $\mathbf{E}_D \psi(x, y) \geq 4\epsilon^2$. In either case, $\mathbf{E}_D \psi(x, y) - 2\mathbf{E}_S \psi(x, y) \leq 4\epsilon^2$. □

The last few results have been presented in terms of general classes of functions. For the remainder of this section we return to considering classes of binary classifiers.

For a class $\mathcal{F}$ of classifiers $f : X \to \{\pm 1\}$, the growth function of $\mathcal{F}$ measures the maximum possible number of unique classifications of a set of examples of size m that can be made using classifiers from $\mathcal{F}$. Clearly, for all m, $\Pi_\mathcal{F}(m) \leq |\mathcal{F}|$ if $\mathcal{F}$ is finite and $\Pi_\mathcal{F}(m) \leq 2^m$. If classifiers from $\mathcal{F}$ can produce all possible classifications of an example set $S \subseteq X$ (i.e., $|\mathcal{F}_{|S}| = 2^{|S|}$), then we say that $\mathcal{F}$ *shatters* S.

Definition 1.2 (VC Dimension)
The VC dimension $\mathrm{VCdim}(\mathcal{F})$ *of a class $\mathcal{F}$ of classifiers $f : X \to \{\pm 1\}$ is defined by*

$$\mathrm{VCdim}(\mathcal{F}) = \max\{|S| : S \subseteq X \text{ and } \mathcal{F} \text{ shatters } S\}$$

or ∞ if no maximum exists.

The VC dimension is the largest value of m for which $\Pi_{\mathcal{F}}(m) = 2^m$. That is, the size of the largest set of examples which can be arbitrarily classified using classifiers from $\mathcal{F}$. Clearly, if $\mathcal{F}$ is finite then $\text{VCdim}(\mathcal{F}) \leq \log_2(|\mathcal{F}|)$. Since the VC dimension can be defined in terms of the growth function, these two quantities are clearly related. However, the following result gives another relationship between the growth function and the VC dimension. This result, independently discovered by Vapnik and Chervonenkis [70], Sauer [59], and Shelah [64], is generally referred to as Sauer's Lemma.

Lemma 1.4 (Sauer's Lemma)
Let $d = \text{VCdim}(\mathcal{F})$, *then*

$$\Pi_{\mathcal{F}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} < \left(\frac{em}{d}\right)^d,$$

where the second inequality holds for $m \geq d$.

The following result is the main theoretical motivation for finding classifiers which minimize training error. It provides a high probability upper bound on the generalization error of *any* binary classifier in a class $\mathcal{F}$ in terms of the classifier's training error and a complexity term which depends on the growth function of the class $\mathcal{F}$ and decreases with the training set size m.

Theorem 1.5 (Vapnik and Chervonenkis [70])
There exists a constant κ *such that with probability at least* $1 - \delta$ *over the random choice of* S *according to* D^m, *every* $f \in \mathcal{F}$ *satisfies*

$$\mathbf{P}_D(f(x) \neq y) \leq \mathbf{P}_S(f(x) \neq y) + \kappa\sqrt{\frac{\ln \Pi_{\mathcal{F}}(2m) + \ln(1/\delta)}{m}}.$$

Proof

This result can be obtained directly from Lemma 1.1 by choosing $\Psi = \{1(f(x) \neq y) : f \in \mathcal{F}\}$, where, $1(\cdot)$ denotes the indicator function which gives 1 when $f(x) \neq y$ and 0 otherwise. After noticing that $\Pi_\Psi = \Pi_{\mathcal{F}}$, the result is obtained by setting the upper bound in Lemma 1.1 to be δ and solving for ϵ. □

By applying Sauer's Lemma to bound the growth function in terms of the VC dimension we immediately obtain the following result involving the VC dimension of $\mathcal{F}$ rather than the growth function of $\mathcal{F}$.

Theorem 1.6 (Vapnik and Chervonenkis [70])
There exists a constant κ *such that with probability at least* $1 - \delta$ *over the random choice*

of S according to D^m, every $f \in \mathcal{F}$ with $\mathrm{VCdim}(\mathcal{F}) = d$ *satisfies*

$$\mathbf{P}_D(f(x) \neq y) \leq \mathbf{P}_S(f(x) \neq y) + \kappa\sqrt{\frac{d\ln(m/d) + \ln(1/\delta)}{m}},$$

for $m \geq d/2$.

Although these results hold for any value of the training set size m, the practical performance exhibited by classification algorithms is generally much better than these theoretical results would suggest. However, these results are asymptotically close to the best possible since there are lower bounds of the same form [2, 21, 65].

One reason for this disparity between theory and practice is the generality of the results. Allowing for *any* distribution generating examples means that these results are necessarily considering worst-case distributions which may be unlikely to occur in practice. Another reason is that this theory ignores the important fact that many classifiers produce real values which are then thresholded to generate a classification. Often, the real-valued output can be interpreted as a measure of confidence in classification. Classifiers of this type include neural networks, support vector machines and combined classifiers produced by voting methods. This notion of 'classification confidence' is formalized by the *margin* of an example.

Definition 1.3 (Margin)
For a binary classifier of the form $\mathrm{sign}(f(x))$, *where* $f : X \to \mathbb{R}$, *the* margin *of an example* (x, y) *is defined to be* $yf(x)$.

The *sign function* $\mathrm{sign} : \mathbb{R} \to \{\pm 1\}$ maps $a \in \mathbb{R}$ to 1 if and only if $a > 0$. It is easy to see that an example is correctly classified if $yf(x) > 0$ and that if we interpret the magnitude of f as 'confidence' in classification then the larger the margin the more confident the classification.

Vapnik [67] first suggested that the margins of the training examples and the generalization error were related. For classifiers which can be represented as thresholded real-valued functions, theoretical results have since been given by various authors giving bounds on the generalization error in terms of the margins of the training examples. Results for support vector machines [63], neural networks [5] and voted classifiers [60] give bounds which improve on those which can be derived from VC theory when most of the training examples are classified with a large margin.

In the analysis used in all of these results, if a thresholded real-valued function classifies most training examples with a large margin, then we can approximate this classifier using a function from a simpler class (and the larger the margin, the simpler the class). If we assume that $f(x)$ is continuous both in x and in the parameters that determine f then classifiers which exhibit large margins on the training examples are more 'reliable' in two

senses :

- Slightly perturbing a training example with a large margin is unlikely to cause a change in its classification.
- Slightly perturbing the classifier is unlikely to change the classification of training examples with large margins.

Intuitively, a classifier which is more reliable on training examples is more likely to generalize well. In the following section we will see how formal analysis in terms of margins can be applied to voted classifiers.

1.1.3 Voted classifiers

Much recent work in machine learning has been concerned with the idea of combining many classifiers in order to improve performance. One way in which classifiers can be combined is by a weighted majority vote. In the binary case, a voted classifier is generally of the form $\text{sign}(F(x))$, where

$$F(x) = \sum_{t=1}^{T} w_t f_t(x) \quad \text{and} \quad f_t : X \to \{\pm 1\}.$$

Each classifier in the combination is called a *base classifier* and is restricted to come from some fixed class $\mathcal{F}$. The class of linear combinations of T classifiers from $\mathcal{F}$ will be denoted by $\text{lin}_T(\mathcal{F})$, while $\text{lin}(\mathcal{F})$ will denote the class of linear combinations of an arbitrary (possibly countably infinite) number of classifiers from $\mathcal{F}$. Usually, we will also make the restriction that the combination be convex. That is,

$$\sum_{t=1}^{T} w_t = 1 \quad \text{and} \quad w_t \geq 0.$$

The class of convex combinations of T classifiers from $\mathcal{F}$ will be denoted by $\text{co}_T(\mathcal{F})$, while $\text{co}(\mathcal{F})$ will denote the class of convex combinations of an arbitrary (again, possibly countably infinite) number of classifiers from $\mathcal{F}$.

Nearly all of the algorithms for producing voted classifiers (hereafter referred to as *voting methods*) iteratively construct the combination one classifier at a time. As such we will denote the combination of the first t classifiers by F_t, while the final combination of T classifiers will simply be denoted by F.

The remainder of this section is devoted to describing the most popular and successful voting method, AdaBoost.

AdaBoost

Freund and Schapire's AdaBoost algorithm [35] is arguably one of the most important developments in practical machine learning in the past decade. Many studies [9, 22, 25, 34, 47, 55, 62] have demonstrated that AdaBoost can produce extremely accurate classifiers from base classifiers as simple as decision stumps or as complex as neural networks or decision trees. AdaBoost was originally developed as a method for solving the 'boosting' problem first posed by Kearns and Valiant [43]. This problem essentially asks whether a 'weak' learning algorithm (which performs only slightly better than random guessing) can be 'boosted' into a 'strong' learning algorithm (which performs arbitrarily well) within Valiant's PAC learning model [66].

Pseudo-code for AdaBoost is shown in Algorithm 1.1. AdaBoost iteratively combines base classifiers from some class $\mathcal{F}$ by repeatedly calling a *base learner* $\mathcal{L}$ which returns classifiers from $\mathcal{F}$. The goal of the base learner is to return a classifier with small weighted error relative to a distribution D over training examples. That is, $\mathcal{L}$ should choose a classifier f from $\mathcal{F}$ such that

$$\sum_{i:\, f(x_i) \neq y_i} D(i)$$

is small. AdaBoost starts with a uniform distribution over training examples and in subsequent rounds adjusts the distribution to emphasize examples which are misclassified by the classifiers returned by $\mathcal{L}$. The weight of the returned classifier in the combination is based upon its weighted error according to D.

Freund and Schapire proved the following result bounding the training error of classifiers produced by AdaBoost in terms of the weighted errors of the base classifiers at each round.

Theorem 1.7 (Freund and Schapire [35])
If AdaBoost generates a combined classifier F with each base classifier f_t having weighted training error ϵ_t then,

$$P_S(\text{sign}(F(x)) \neq y) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}. \tag{1.1}$$

If $\epsilon_t \leq \frac{1}{2} - \gamma$, for some fixed constant γ, then it is easy to see that the product term in (1.1) is less than one. So, provided that the base classifiers perform a fixed amount better than random guessing (relative to the weighted training set) at each round, the training error will decrease to zero at an exponential rate as boosting proceeds.

Although this result gives impressive performance guarantees for the training error, it says nothing about the generalization error. However, Theorem 1.6 can be directly applied

Algorithm 1.1: AdaBoost [35]

Require :

- A class of base classifiers $\mathcal{F}$ containing functions $f : X \to \{\pm 1\}$.
- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in X \times \{\pm 1\}$.
- A base learner $\mathcal{L}(S, D)$ that accepts a training set S and a distribution D on the training set and returns base classifiers $f \in \mathcal{F}$ with small weighted error $\sum_{i:\, f(x_i) \neq y_i} D(i)$.

let $F_0(x) := 0$
let $D_1(i) := 1/m$ for $i = 1, \ldots, m$
for $t := 1$ **to** T
  let $f_t := \mathcal{L}(S, D_t)$
  let $\epsilon_t := \sum_{i:\, f_t(x_i) \neq y_i} D_t(i)$
  if $\epsilon_t \geq \frac{1}{2}$ **then**
    return F_{t-1}
  let $w_t := \frac{1}{2} \ln\big((1 - \epsilon_t)/\epsilon_t\big)$
  let $F_t := F_{t-1} + w_t f_t$
  let $Z_t := 2\sqrt{\epsilon_t(1 - \epsilon_t)}$
  let $D_{t+1}(i) := D_t(i) \exp(-y_i w_t f_t(x_i))/Z_t$ for $i = 1, \ldots, m$
return F_T

after bounding the VC dimension of the class of combinations of classifiers in terms of the VC dimension of the class of base classifiers. Freund and Schapire [35] apply this technique to prove the following result.

Theorem 1.8 (Freund and Schapire [35])
There exists a constant κ such that with probability at least $1 - \delta$ over the random choice of S according to D^m, every $F \in \operatorname{lin}_T(\mathcal{F})$ satisfies

$$P_D(\operatorname{sign}(F(x)) \neq y) \leq P_S(\operatorname{sign}(F(x)) \neq y) + \kappa\sqrt{\frac{d \ln(m/d) + \ln(1/\delta)}{m}},$$

where $d = T \ln T \operatorname{VCdim}(\mathcal{F})$.

The important thing to notice about this result is that the term inside the square root grows at least linearly with the number of classifiers in the combination. So the generalization error should increase once the classifier achieves zero error on the training set (which is guaranteed to happen quickly by Theorem 1.7). However, in practice this was often *not* seen to be the case [25, 55, 60]. This behaviour is illustrated in Figure 1.1, which compares the training error to the error on an independent test set for AdaBoost (using decision trees as base classifiers) on the synthetic data set `twonorm` (see [14] for a

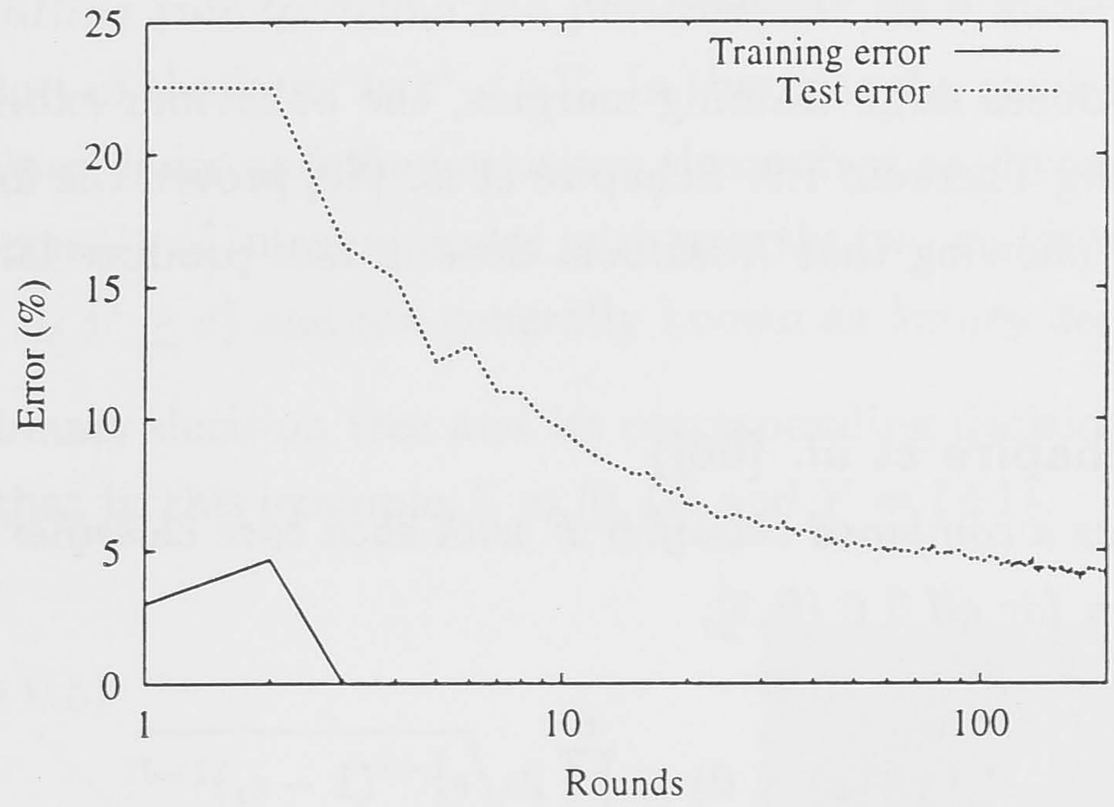


Figure 1.1: AdaBoost training and test error over 200 boosting rounds for the `twonorm` data set using decision trees as base classifiers. Notice that the test error continues to decrease even after the training error has converged to zero.

definition of this data set). Notice that the test error of AdaBoost decreases significantly throughout the entire run, even though the training error converges to zero after the first few rounds.

Based upon earlier work due to Bartlett [5], Schapire *et al.* [60] gave the first theoretical justification of this behaviour. By applying the same techniques Bartlett used to analyze neural networks in terms of the margin they proved the following result.

Theorem 1.9 (Schapire *et al.* [60])
There exists a constant κ such that with probability at least $1-\delta$ over the random choice of S according to D^m, every $F \in \mathrm{co}(\mathcal{F})$ and $\theta \in (0,1]$ satisfies

$$P_D(\mathrm{sign}(F(x)) \neq y) \leq P_S(yF(x) \leq \theta) + \kappa\sqrt{\frac{1}{m}\left(\frac{d\ln^2(m/d)}{\theta^2} + \ln(1/\delta)\right)},$$

where $d = \mathrm{VCdim}(\mathcal{F})$.

This result, like Theorem 1.8, bounds the generalization error using two terms. The first term, rather than depending on the training error, depends on the proportion of the training set with margin less than some value θ. The second term again depends on the VC dimension of the class of base classifiers, but this time with a penalty on the size of θ (which prevents us taking θ too small). Note, however, that the second term has *no* dependence on T, the number of classifiers in the combination. The bound holds for any choice of θ and to obtain the best result θ should be chosen to best balance the two terms. Intuitively, if the combined classifier attains large margins on the training set then $P_S(yF(x) \leq \theta)$ won't be much larger than $P_S(yF(x) \leq 0)$, even for large values of θ. Using a large value of θ means that the second term is roughly $\sqrt{\mathrm{VCdim}(\mathcal{F})/m}$ rather

than the $\sqrt{T\,\mathrm{VCdim}(\mathcal{F})/m}$ of Theorem 1.8.

So, if AdaBoost produces large training margins, the behaviour exhibited in Figure 1.1 can be explained using Theorem 1.9. Schapire *et al.* [60] proved the following generalization of Theorem 1.7 showing that AdaBoost does in fact produce large margins on the training set.

Theorem 1.10 (Schapire *et al.* [60])
If AdaBoost generates a combined classifier F with each base classifier f_t having weighted training error ϵ_t then for all $\theta \in (0, 1]$,

$$P_S(yF(x) \leq \theta) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}}. \tag{1.2}$$

If $\epsilon_t < \frac{1}{2} - \theta - \gamma$, for some fixed constant γ, then it can be shown that the product term in (1.2) is less than one and thus the proportion of training examples with margin larger than θ decreases to zero at an exponential rate as boosting proceeds.

1.1.4 Decision tree classifiers

Decision tree classifiers have become extremely popular in recent years, particularly in commercial implementations of classification systems (such as Rulequest Research's C4.5 and C5.0, Salford Systems' CART, SGI's MineSet and IBM's Intelligent Miner). This popularity is due largely to three factors, efficiency, interpretability and accuracy.

A *decision tree* classifier partitions the measurement space X (which we assume to be a subset of $\mathbb{R}^n$) into regions, each of which is associated with a classification label. A decision tree consists of a root node, a set of internal nodes (including the root node), a set of leaf (or terminal) nodes, and a set of directed edges connecting nodes.

Each node in the tree is associated with a region which is a subset of $\mathbb{R}^n$. Every internal node is connected by edges to one or more child nodes which satisfy the following property. If an internal node is associated with the region A and its k child nodes are associated with the regions $A_1, \ldots, A_k$ respectively, then $A = A_1 \cup \cdots \cup A_k$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$. That is, the child nodes' regions form a partition of their parent's region. The root node is associated with the entire measurement space X. Thus, proceeding from the root of the tree, each node recursively subdivides the measurement space X into successively smaller regions. By associating each leaf node with a classification label every point in X is associated with a unique classification. The depth of a leaf is the number of edges from the root node to the leaf.

In order to make such classifiers practical, the process of finding the region into which an example $x = (x^1, \ldots, x^n)$ falls should be fast. Typically, this is accomplished by

using a simple *splitting rule* to define the partitions at each node. The most common splitting rule is a test of the form 'Is $x^i < a$?'. In this case the partitions are formed using axis orthogonal hyperplanes and the partitions themselves are hyperrectangles. Decision trees of this type consist of internal nodes with exactly two children (one corresponding to $x^i < a$ and one to $x^i \geq a$) and are generally known as *binary decision trees.*

An example of a binary decision tree and its corresponding decision regions is shown in Figure 1.2. Note that in this example $X = [0, 1]^2$ and $Y = \{\pm 1\}$.

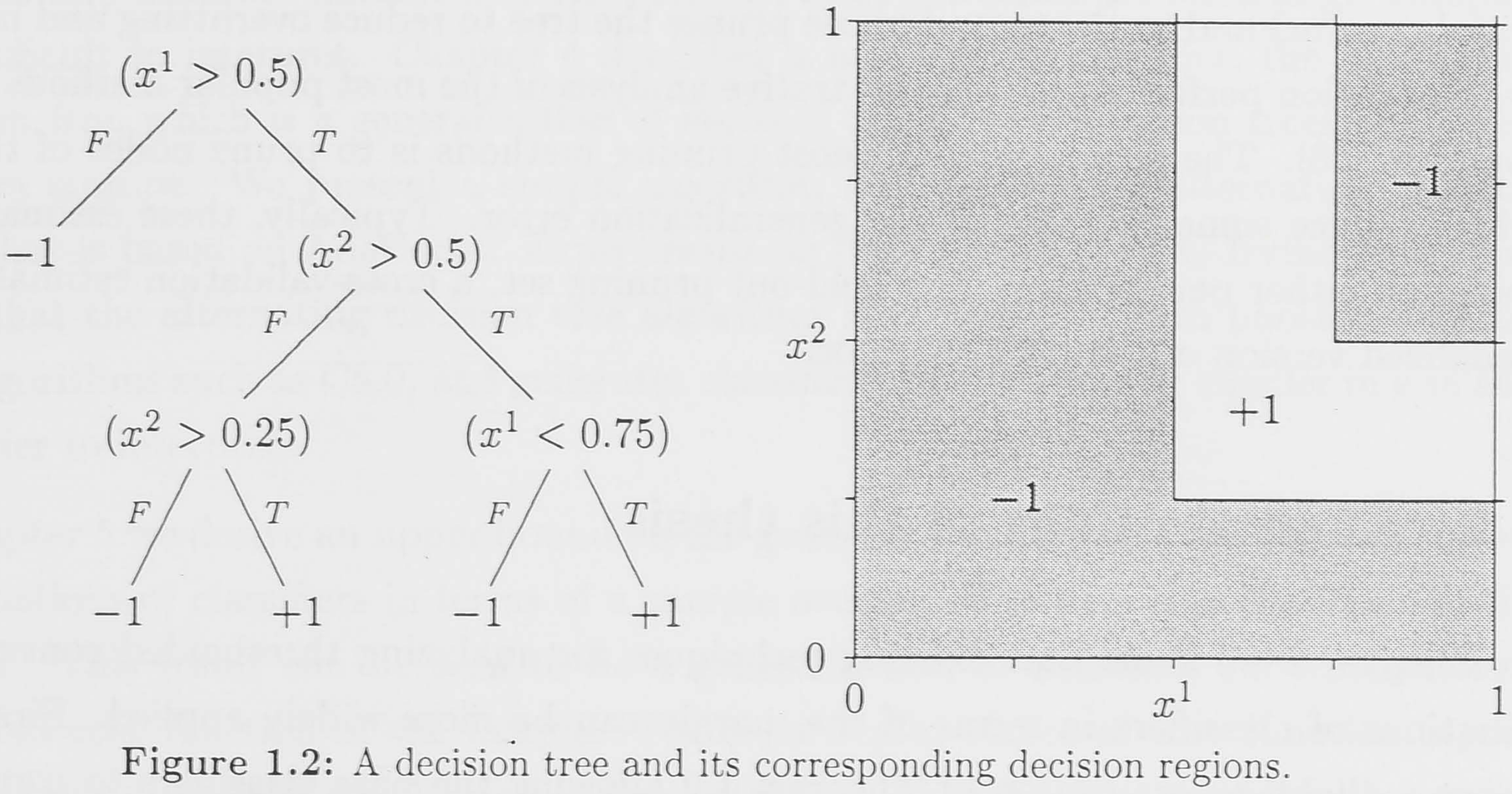


Figure 1.2: A decision tree and its corresponding decision regions.

In many of the experimental results presented in this thesis we will make use of a restricted form of the binary decision tree, the *decision stump.* This is simply a binary decision tree consisting of a single internal node with a split of the form 'Is $x^i < a$?' and two child nodes corresponding to $x^i < a$ and $x^i \geq a$. Since such a decision tree partitions X using a single axis orthogonal hyperplane, decision stumps are sometimes referred to as *axis-orthogonal hyperplane classifiers.*

Returning to general decision trees, it is often the case that some measurements lie in a discrete set rather than in a real interval. In this case associating partitions with each possible value (or subsets of values) of the discrete measurement can be more useful. In this case internal nodes can have more than two children. Another less common splitting rule is a test of the form 'Is $b_1 x^1 + \cdots + b_n x^n < a$?'. Partitions are then arbitrary convex polyhedra formed using hyperplanes. Of course, such splits are more expensive to evaluate than axis orthogonal hyperplanes.

There is a vast body of literature on the design of decision trees and a description of available methods and their advantages and disadvantages is beyond the scope of this thesis. However, two seminal works should be mentioned. In [16], Breiman *et al.* describe the CART (Classification And Regression Trees) algorithm, and in [53], Quinlan describes the ID3 algorithm which has since evolved into the C4.5 [54] and C5.0 algorithms (the latter of which combines a version of AdaBoost with C4.5).

All of these algorithms use *top-down induction* to construct decision trees. Top-down induction is a two-phase procedure. In the first phase, starting from a root node, a *splitting criterion* is applied recursively. The criterion determines how a node's corresponding region should be partitioned. Reviews of the most popular splitting criteria can be found in [18, 30, 51]. Since this recursive splitting process is generally applied until the leaves of the tree contain only examples of a single class (or no further useful splits are possible), the order in which nodes are split is irrelevant.

The tree constructed from the first phase is likely to overfit (as it will typically have zero training error) and so the second phase prunes the tree to reduce overfitting and improve generalization performance. A comparative analysis of the most popular methods can be found in [28]. The general idea of most pruning methods is to prune nodes of the tree which reduce some estimate of the generalization error. Typically, these estimates are based on either performance on a held-out pruning set, a cross-validation estimate, or a penalized version of the training error.

1.2 Contributions of this thesis

In Chapter 2 we show that existing techniques for analyzing thresholded convex combinations of classifiers in terms of the margin can be more widely applied. Firstly, we prove a slight generalization of Theorem 1.9 allowing the base classifiers to come from several different classes rather than a single class. Then, by representing a decision tree as a thresholded convex combination of leaf functions, we derive an upper bound on the generalization error of decision trees. Existing bounds derived from VC theory indicate that the complexity of a decision tree depends on the total number of leaves in the tree. Our bound uses a different notion of complexity involving the distribution of training examples over leaves. For decision trees which classify most of the training examples using only a few leaves, our bound can be considerably smaller than those suggested by VC theory. Experimental results on a selection of data sets from the UC Irvine repository demonstrate that decision trees generated by algorithms like C4.5 often use small numbers of leaves to classify most examples.

In Chapter 3 we derive an upper bound on the generalization error of classifiers which can be represented as thresholded convex combinations of functions which are themselves thresholded convex combinations. Such classifiers include single hidden-layer threshold networks and voted combinations of decision trees. The derived bound depends on the proportion of training examples with margin less than some threshold and the average complexity of the combined classifiers (where the average is over the weights assigned to each classifier in the convex combination). The complexity of the individual classifiers in the combination depends on their closeness to threshold. Following the representation of decision trees described in the previous chapter we apply this result to bound the

generalization error of combinations of decision trees in terms of the margin of the classifier and the average complexity of the individual decision trees. The complexity of each decision tree depends on a similar notion of complexity to that described in the previous chapter. Previous bounds depend on the complexity of the most complex decision tree in the combination, where the complexity of each decision tree depends on the total number of leaves.

Combinations of decision trees constructed using voting methods have been shown to produce very accurate classifiers. Unfortunately, these classifiers are often large, complex and difficult to interpret. Chapter 4 describes a new type of classifier, the *alternating decision tree*, which is a generalization of decision trees, voted decision trees and voted decision stumps. We present a simple algorithm for constructing alternating decision trees that is based on AdaBoost. Experiments on data sets in the UC Irvine repository show that the alternating decision tree algorithm is competitive with boosted decision tree algorithms such as C5.0, and generates classifiers that are usually smaller in size and far easier to interpret.

In Chapter 5 we derive an upper bound on the generalization error of thresholded convex combinations of classifiers in terms of a sample average of general cost functions of the margin. This result can be seen to be a generalization of Theorem 1.9 which uses a threshold cost function of the margin. For a particular choice of cost function, upper bounds can be obtained which improve on existing results and can be considerably smaller than those suggested by VC theory.

In Chapter 6 we present a new algorithm, *DOOM*, for directly optimizing a piecewise-linear family of cost functions satisfying the conditions of the main result of the previous chapter. Experiments on several of the data sets in the UC Irvine repository are presented in which AdaBoost was used to generate a set of base classifiers and then DOOM was used to find the optimal convex combination of those classifiers. Generally, the convex combination generated by DOOM improves on the test error of AdaBoost's convex combination. In many cases DOOM achieves these lower test errors by sacrificing training error, in the interests of reducing the new cost function. Several authors [14, 15, 38] have interpreted Theorem 1.9 to mean that the success of voting methods is due to their ability to maximize the value of the minimum training margin (provided that the complexity of the class of base classifiers is fixed). Graphs of the training margin distributions generated using DOOM suggest that the value of the minimum margin is not a critical factor in determining generalization performance.

In Chapter 7 we describe a general class of gradient descent algorithms, *AnyBoost*, for choosing linear combinations of elements of an inner product space so as to minimize some cost functional. Each component of the linear combination is chosen to maximize a certain inner product. We provide convergence results for this class of algorithms with a convex cost functional, showing that with a particular choice of the step size, if each

component of the linear combination maximizes the inner product then the algorithm converges to the global minimum of the cost functional.

We then examine a special case of AnyBoost, *MarginBoost*, for choosing a combination of classifiers to optimize the sample average of a cost function of the margin. MarginBoost performs gradient descent in function space, at each iteration choosing a base classifier to include in the combination so as to maximally reduce the cost function. The idea of performing gradient descent in function space in this way is due to Breiman [14]. It turns out that, as in AdaBoost, the choice of the base classifier corresponds to a minimization problem involving weighted classification error. That is, for a certain weighting of the training data, the base classifier learning algorithm attempts to return a classifier that minimizes the weight of misclassified training examples.

We then show that this general class of algorithms includes as special cases a number of popular and successful voting methods, including Freund and Schapire's AdaBoost [35], Schapire and Singer's extension of AdaBoost to combinations of real-valued functions [61], Breiman's ARC-X4 [15] and Friedman, Hastie and Tibshirani's LogitBoost [37]. That is, all of these algorithms implicitly minimize some cost function of the margin by gradient descent.

In Chapter 8 we examine a restriction of the MarginBoost algorithm, *DOOM II*, designed to iteratively construct a *convex* combination of classifiers which minimize the sample average of a cost function of the margin which is qualitatively similar to the theoretically motivated cost functions derived in Chapter 5. Experiments on several data sets from the UC Irvine repository demonstrate that DOOM II generally outperforms AdaBoost, especially in high noise situations. Graphs of the training margin distributions generated by DOOM II verify that the algorithm is willing to 'give up' on examples which are too hard in order to avoid overfitting. We also show that the overfitting behavior exhibited by AdaBoost can be quantified in terms of our proposed cost function.

We present conclusions and outline potential future work in Chapter 9.

CHAPTER

2

Threshold margin bounds for decision trees

■ ■ ■

For decision tree classifiers, results from VC theory suggest that the amount of training data should grow at least linearly with the size of the tree [27, 29]. However, recent empirical results suggest that this is not always necessary [52, 73]. For example, Murphy and Pazzani [52] have observed that the generalization error is not even always a monotonically increasing function of the tree size.

To see why the size of a tree is not always a good measure of its complexity, consider two trees, A with T_A leaves and B with T_B leaves, where $T_B \ll T_A$. Thus, size-wise, A is more complex than B. However, in a case where most of the classification in A is carried out by very few leaves and the classification in B is equally distributed over the leaves, intuition suggests that A is actually much simpler than B, since tree A can be approximated well by a small tree with few leaves.

In this chapter, we formalize this intuition by deriving upper bounds on the generalization error of decision trees in terms of a new complexity measure which depends on the distribution on the leaves that is induced by the set of training examples. These bounds are qualitatively different from the bounds obtained from VC theory, which depend on the total number of leaves in the tree, and can be considerably smaller.

Our results build on previous results which give bounds on the generalization error of convex combinations of classifiers in terms of the margin. In fact, in Section 2.1, we derive a refinement of Theorem 1.9 for voted combinations of classifiers, in which the base classifiers can come from several classes of different complexity, and the VC dimension of the class of base classifiers is replaced by the average (over the voting weights) of the VC dimensions of these classes.

In Section 2.2, we extend these results to decision trees by considering a decision tree as a thresholded convex combination of the *leaf functions*

(the Boolean functions that specify, for a given leaf, which examples reach that leaf). We can then apply the generalization error bounds for such classifiers. The resulting bounds depend on the distribution of training examples over the tree's leaves. In Section 2.3, we show that, by making certain assumptions on the form of this distribution, our result can be considerably smaller than the results given by VC theory. Finally, in Section 2.4, we demonstrate that these assumptions often hold in practice by examining the decision trees generated by C4.5 for a selection of data sets from the UC Irvine repository.

2.1 Combinations of classifiers

Let us consider a voted combination of classifiers defined by $\text{sign}(F(x))$, where

$$F(x) = \sum_{t=1}^{T} w_t f_t(x) \tag{2.1}$$

with $\sum_{t=1}^{T} w_t = 1$, $w_t \geq 0$ and $f_t : X \to \{\pm 1\}$.

Theorem 1.9 gives an upper bound on the generalization error of such classifiers in terms of the proportion of training examples with margin less than some threshold and a complexity term which involves the VC dimension of the class of base classifiers. In using this bound for a particular convex combination of classifiers the complexity term necessarily depends on the complexity of the most complex classifier, even if this classifier has a low voting weight and hence little effect on the combination. This is due to the fact that all of the base classifiers f_t in the convex combination F are elements of a single class $\mathcal{F}$ with bounded VC dimension. The following theorem generalizes this result to the case in which the base classifiers may be chosen from any of r classes, $\mathcal{F}_1, \ldots, \mathcal{F}_r$, which can have different VC dimensions. That is, $f_t \in \mathcal{F}_{n_t}$ for $n_t \in \{1, \ldots, r\}$. The bound in the theorem depends on the proportion of training examples with margin less than some threshold and on a complexity term which involves the *average* VC dimension of the classes of base classifiers (where the average is in terms of the voting weights assigned to each of the base classifiers in the convex combination).

Theorem 2.1

There exists a constant κ such that with probability at least $1 - \delta$ over the random choice of S according to D^m, every $F \in \text{co}\left(\bigcup_{n=1}^{r} \mathcal{F}_n\right)$, $0 < \theta \leq 1$ satisfies

$$\begin{aligned} \mathbf{P}_D(yF(x) \leq 0) \quad \leq \quad & \mathbf{P}_S(yF(x) \leq \theta) \\ & + \kappa \sqrt{\frac{1}{m}} \left(\frac{1}{\theta^2} (\nu + \ln r) \ln \left(m\theta^2 / \nu \right) + \ln(1/\delta) \right)^{1/2} \end{aligned}$$

where $\nu = \sum_{t=1}^{T} w_t \ln \Pi_{\mathcal{F}_{n_t}}(2m)$.

By applying Sauer's Lemma to bound the growth function in terms of the VC dimension we immediately obtain the following result involving the VC dimension rather than the growth function.

Theorem 2.2

There exists a constant κ such that with probability at least $1-\delta$ over the random choice of S according to D^m, every $F \in \mathrm{co}\left(\bigcup_{n=1}^{r} \mathcal{F}_n\right)$, $0 < \theta \leq 1$ satisfies

$$\begin{aligned}\mathbf{P}_D(yF(x) \leq 0) \quad &\leq \quad \mathbf{P}_S(yF(x) \leq \theta) \\ &\quad + \kappa\sqrt{\frac{1}{m}}\left(\frac{1}{\theta^2}(\mu \ln m + \ln r)\ln\left(m\theta^2/\mu\right) + \ln(1/\delta)\right)^{1/2}\end{aligned}$$

where $\mu = \sum_{t=1}^{T} w_t \,\mathrm{VCdim}(\mathcal{F}_{n_t})$.

Proof (Theorem 2.1)

The proof closely follows the proof of Theorem 1.9 (which is given in [60]).

We define an approximating class for $\mathrm{co}(\bigcup_{n=1}^{r} \mathcal{F}_n)$ by

$$\mathcal{G} = \bigcup_{l} \mathcal{G}_l \quad \text{where} \quad \mathcal{G}_l = \left\{ \tfrac{1}{N}\sum_{i=1}^{N} g_i : g_i \in \mathcal{F}_{l_i} \right\} \quad \text{and} \quad l = (l_1, \ldots, l_N).$$

Given $F(x) = \sum_t w_t f_t(x)$ where $f_t \in \mathcal{F}_{n_t}$ we define a distribution over $\mathcal{G}$ via $\mathbf{P}(g_i = f_t) = w_t$. Note that if $g_i = f_t$ then $l_i = n_t$. The probability of randomly choosing a $G \in \mathcal{G}$ according to this distribution which satisfies a predicate A will be denoted by $\mathbf{P}_G(A)$ while the expected value over random choice of G will be denoted by $\mathbf{E}_G$.

In order to reduce the inequality in the theorem to a standard structural risk minimization result we require the following lemma. Here, $1(A)$ denotes the indicator function of a predicate A.

Lemma 2.3

For all $F \in \mathrm{co}(\bigcup_{n=1}^{r} \mathcal{F}_n)$, $N > 0$, $x \in X$, $y \in \{\pm 1\}$ and $\theta \in (0,1]$

$$(a) \quad 1(yF(x) \leq 0) \leq \mathbf{E}_G 1(yG(x) \leq \theta/2) + e^{-N\theta^2/8}$$

$$(b) \quad \mathbf{E}_G 1(yG(x) \leq \theta/2) \leq 1(yF(x) \leq \theta) + e^{-N\theta^2/8}$$

Proof

(a) Clearly the inequality holds for $yF(x) > 0$. If $yF(x) \leq 0$ then we need

$$1 - \mathbf{P}_G(yG(x) \leq \theta/2) \leq e^{-N\theta^2/8}.$$

Since $yF(x) \leq 0$, $1 - \mathbf{P}_G(yG(x) \leq \theta/2) \leq \mathbf{P}_G(yG(x) - yF(x) > \theta/2)$, which can in turn be bounded above by $e^{-N\theta^2/8}$ using Hoeffding's inequality.

(b) Clearly the inequality holds for $yF(x) \leq \theta$. If $yF(x) > \theta$ then we need

$$\mathbf{P}_G(yG(x) \leq \theta/2) \leq e^{-N\theta^2/8}.$$

Since $yF(x) > \theta$, $\mathbf{P}_G(yG(x) \leq \theta/2) \leq \mathbf{P}_G(yF(x) - yG(x) > \theta/2)$, which can in turn be bounded above by $e^{-N\theta^2/8}$ using Hoeffding's inequality. □

We are now going to bound $\mathbf{P}_D(yF(x) \leq 0)$ in terms of $\mathbf{P}_S(yF(x) \leq 0)$ and a complexity term c. By suitable choice of c we will obtain the result.

$$\mathbf{P}_{D^m}\left(\exists F \in \mathrm{co}\left(\bigcup_{n=1}^{r} \mathcal{F}_n\right), N > 0, \theta \in (0,1] : \right.$$
$$\left. \mathbf{P}_D(yF(x) \leq 0) > \mathbf{P}_S(yF(x) \leq \theta) + \mathbf{E}_G c + 2e^{-N\theta^2/8}\right)$$
$$\leq \mathbf{P}_{D^m}(\exists F, N, \theta : \mathbf{E}_G\mathbf{E}_D 1(yG(x) \leq \theta/2) > \mathbf{P}_S(yF(x) \leq \theta) + \mathbf{E}_G c + e^{-N\theta^2/8})$$
[by applying Lemma 2.3(a)]
$$\leq \mathbf{P}_{D^m}(\exists F, N, \theta : \mathbf{E}_G\mathbf{E}_D 1(yG(x) \leq \theta/2) > \mathbf{E}_G\mathbf{E}_S(yG(x) \leq \theta/2) + \mathbf{E}_G c)$$
[by applying Lemma 2.3(b)]
$$\leq \mathbf{P}_{D^m}(\exists N, \theta, G \in \mathcal{G} : \mathbf{P}_D(yG(x) \leq \theta/2) > \mathbf{P}_S(yG(x) \leq \theta/2) + c)$$
$$= \mathbf{P}_{D^m}(\exists N, \theta, l, G \in \mathcal{G}_l : \mathbf{P}_D(yG(x) \leq \theta/2) > \mathbf{P}_S(yG(x) \leq \theta/2) + c)$$

This can be bounded above by

$$\sum_{N=1}^{\infty}(N+1)r^N \max_{\theta,l} \mathbf{P}_{D^m}(\exists G \in \mathcal{G}_l : \mathbf{P}_D(yG(x) \leq \theta/2) > \mathbf{P}_S(yG(x) \leq \theta/2) + c) \quad (2.2)$$

by repeated application of the union bound over N, θ and l. Since G is the sum of N $\{\pm 1\}$-valued functions there are only $N+1$ values of θ which make any difference to the probability. Also, each of the N values of l_i are chosen from $\{1, \ldots, r\}$.

To bound this last probability in (2.2) we apply Lemma 1.1 with $\Psi = \{1(yG(x) \leq \theta/2) : G \in \mathcal{G}_l\}$. For this choice of Ψ, $\Pi_\Psi(m)$ is bounded above by the product of the growth functions of the N choices of functions from the $\mathcal{F}_{l_i}$. Hence, (2.2) is bounded above by

$$\sum_{N=1}^{\infty}(N+1)r^N \max_{\theta,l} 4\prod_{i=1}^{N}\Pi_{\mathcal{F}_{l_i}}(2m)e^{-mc^2/8}. \quad (2.3)$$

Now letting

$$c = \sqrt{\frac{8}{m}}\left(\ln(N(N+1)) + \ln(N+1) + N\ln r + \ln 4 + \sum_{i=1}^{N} \ln \Pi_{\mathcal{F}_{l_i}}(2m) + \ln(1/\delta)\right)^{1/2}$$

implies that (2.3) is equal to $\sum_{N=1}^{\infty} \frac{1}{N(N+1)}\delta = \delta$. By application of Jensen's inequality,

$$\mathbf{E}_G c \leq \sqrt{\frac{8}{m}}\Bigg(\ln(N(N+1)) + \ln(N+1) + N\ln r + \ln 4 + N\sum_t w_t \ln \Pi_{\mathcal{F}_{n_t}}(2m) + \ln(1/\delta)\Bigg)^{1/2}.$$

Finally, substituting for

$$N = \left\lceil \frac{4}{\theta_0^2} \ln \frac{m\theta^2}{\nu} \right\rceil,$$

where $\nu = \sum_t w_t \ln \Pi_{\mathcal{F}_{n_t}}(2m)$, and simplifying gives

$$E_G c + 2e^{-N\theta^2/8} \leq \kappa \sqrt{\frac{1}{m}}\left(\frac{1}{\theta^2}(\nu + \ln r)\ln(m\theta^2/\nu) + \ln(1/\delta)\right)^{1/2}$$

for some constant κ. □

2.2 Decision trees

Theorem 2.1 gives bounds on the generalization error of classifiers which can be represented as thresholded convex combinations of binary functions. The key technique we use in the remainder of this chapter is to represent a decision tree as such a convex combination. As we shall see, we have some freedom in choosing the convex coefficients, and this choice affects both the error estimate $\mathbf{P}_S(yF(x) \leq \theta)$ and the average growth function ν. We will then attempt to choose the coefficients and the margin θ so as to optimize the resulting bound on generalization error.

Given a decision tree with T leaves, let $\sigma_t \in \{\pm 1\}$ denote the label associated with leaf t and $L_t(x)$ be the $\{0,1\}$–valued function producing 1 if and only if x reaches leaf t. Then for any sequence of w_t such that $w_t > 0$ and $\sum_{t=1}^{T} w_t = 1$ the output of the tree for an example x can be written as

$$\operatorname{sign}\left(\sum_{t=1}^{T} w_t \sigma_t L_t(x)\right).$$

Now setting $f_t(x) = \sigma_t L_t(x)$ gives that the output of the tree for an example x can be written as $\text{sign}(F(x))$ where $F(x)$ is defined as in (2.1).

Denoting the depth of leaf t by d_t and the maximum depth of leaves in the tree by d we define the average depth $\bar{d}$ of the tree relative to leaf weights w_t by

$$\bar{d} = \sum_{t=1}^{T} w_t d_t.$$

We now define a weighting $\{P_t^*\}$ over leaves via $P_t^* = \mathbf{P}_S(L_t(x) = 1 \wedge \sigma_t = y)$. Thus P_t^* is simply the proportion of training examples which are classified correctly using leaf t. Note that this is not a probability distribution over leaves since we are only considering the examples which are correctly classified.

We are now ready to apply Theorem 2.1. The following result bounds the generalization error of any decision tree in terms of the training error and two complexity terms which depend on the weighting $\{P_t^*\}$, the average depth $\bar{d}$, and the VC dimension of the class $\mathcal{U}$ of node decision functions.

Theorem 2.4

There exists a constant κ such that with probability at least $1-\delta$ over the random choice of S according to D^m, every decision tree represented as $\text{sign}(F)$ as defined above and every $\theta \in (0,1]$ satisfies

$$\begin{aligned}\mathbf{P}_D(\text{sign}(F(x)) \neq y) \quad \leq \quad & \mathbf{P}_S(\text{sign}(F(x)) \neq y) + \sum_{t=1}^{T} P_t^* 1(w_t \leq \theta) \\ & + \kappa \sqrt{\frac{1}{m}} \left(\frac{1}{\theta^2}(\nu \ln m + \ln d) \ln\left(m\theta^2/\nu\right) + \ln(1/\delta) \right)^{1/2}\end{aligned}$$

where $\nu = \bar{d}\,\text{VCdim}(\mathcal{U})$ and w_t is any sequence such that $w_t > 0$ and $\sum_{t=1}^{T} w_t = 1$.

Proof

Setting $r = d$ (recalling that d is the maximum depth of leaves in the tree), for each $n = 1, \ldots, r$ we define the class $\mathcal{F}_n$ by

$$\mathcal{F}_n = \{\sigma L : \sigma \in \{\pm 1\}, L \in \mathcal{B}_n\},$$

where $\mathcal{B}_n$ is the class of leaf functions for leaves up to depth n defined by

$$\mathcal{B}_n = \{L : L = u_1 \wedge u_2 \wedge \cdots \wedge u_v | v \leq w, u_1, \ldots, u_v \in \mathcal{U}\},$$

where $\mathcal{U}$ denotes the class of node decision functions. So we are separating the leaf

functions into d classes according to their depth. Hence,

$$\begin{aligned}\Pi_{\mathcal{F}_n}(m) &= \max\left\{\left|\left\{\left(\sigma L(x_1),\ldots,\sigma L(x_m)\right) : \sigma\in\{\pm 1\}, L\in\mathcal{B}_n\right\}\right| : x_1,\ldots,x_m\in X\right\}\\ &\leq 2\prod_{l=1}^{n}\max\left\{\left|\left\{\left(u(x_1),\ldots,u(x_m)\right) : u\in\mathcal{U}\right\}\right| : x_1,\ldots,x_m\in X\right\}\\ &= 2\left(\Pi_{\mathcal{U}}(m)\right)^n \qquad (2.4)\end{aligned}$$

and thus ν from Theorem 2.1 satisfies

$$\begin{aligned}\nu &= \sum_{t=1}^{T} w_t \ln\left(\Pi_{\mathcal{F}_{n_t}}(2m)\right)\\ &\leq 2\sum_{t=1}^{T} w_t d_t \ln\Pi_{\mathcal{U}}(2m)\\ &= 2\bar{d}\ln\Pi_{\mathcal{U}}(2m)\\ &\leq 2\bar{d}\,\mathrm{VCdim}(\mathcal{U})\ln(2em). \qquad (2.5)\end{aligned}$$

The first inequality follows from (2.4) since d_t denotes the depth of leaf t, while the last inequality follows from application of Sauer's Lemma.

We also note that

$$\begin{aligned}\mathbf{P}_S(yF(x)\leq\theta) &= \mathbf{P}_S\left(y\sum_{t=1}^{T} w_t\sigma_t L_t(x)\leq\theta\right)\\ &\leq \mathbf{P}_S(\mathrm{sign}(F(x))\neq y) + \mathbf{P}_S\left(0<y\sum_{t=1}^{T} w_t\sigma_t L_t(x)\leq\theta\right)\\ &= \mathbf{P}_S(\mathrm{sign}(F(x))\neq y) + \mathbf{P}_S(w_t\leq\theta\wedge\sigma_t=y)\\ &\qquad [\text{where for each } x,\ t \text{ is such that } L_t(x)=1]\\ &= \mathbf{P}_S(\mathrm{sign}(F(x))\neq y) + \sum_{t=1}^{T} P_t^* 1(w_t\leq\theta), \qquad (2.6)\end{aligned}$$

where $1(\cdot)$ denotes the indicator function. The second last equality follows from the fact that for each value of x, $L_t(x)=0$ for all values of t except the one corresponding to the leaf which is used to classify x.

The final result follows after substituting (2.5) and (2.6) in the statement of Theorem 2.1 and simplifying. □

We now consider how this result compares to existing VC bounds for decision trees. After bounding the VC dimension of the class of decision trees (see, for example, Chapter 21 in [20]) we can apply Theorem 1.6 directly to show that roughly (ignoring constants and

log terms),

$$\mathbf{P}_D(F(x) \neq y) \leq \mathbf{P}_S(F(x) \neq y) + \sqrt{\frac{T \operatorname{VCdim}(\mathcal{U})}{m}}. \tag{2.7}$$

In comparison, our result is roughly (again ignoring constants and log terms),

$$\mathbf{P}_D(F(x) \neq y) \leq \mathbf{P}_S(F(x) \neq y) + \sum_{t=1}^{T} P_t^* 1(w_t \leq \theta) + \sqrt{\frac{\bar{d} \operatorname{VCdim}(\mathcal{U})}{m\theta^2}}.$$

In this result we want to choose θ as large as possible (to minimize the third term on the right hand side) while ensuring that $\sum_{t=1}^{T} P_t^* 1(w_t \leq \theta)$ is close to zero. If we assume that the decision tree is not too asymmetric, the depth of the tree is roughly the logarithm of the number of leaves. In this case, we are comparing T in the VC bound to $(\ln T)/\theta^2$ in our result. If the decision tree uses a small number of leaves to classify most training examples (i.e., the weighting $\{P_t^*\}$ is skewed) then it will be possible to choose a large value of θ (as $\sum_{t=1}^{T} P_t^* 1(w_t \leq \theta)$ will be small even for large θ) and our result will be significantly smaller than the VC bound.

2.3 Optimizing the bound

Different choices of the w_ts and θ in Theorem 2.4 will yield different upper bounds on the generalization error of the tree. We will now see that, by making assumptions on the form of the weighting $\{P_t^*\}$, we can actually optimize over choice of the w_ts and θ to minimize the resulting upper bound.

The bound in Theorem 2.4 is bounded above by

$$\mathbf{P}_D(\operatorname{sign}(F(x)) \neq y) \leq \mathbf{P}_S(\operatorname{sign}(F(x)) \neq y) + \sum_{t=1}^{T} P_t^* 1(w_t \leq \theta) + \frac{1}{\theta}\sqrt{\frac{B}{m}}, \tag{2.8}$$

where $B = \kappa\left((\nu \ln m + \ln d) \ln\left(m\theta^2/\nu\right) + \ln(1/\delta)\right)$ for some constant κ.

Notice that choosing the w_ts amounts to choosing which of the P_t^*s are included in the sum in the second term on the right hand side of (2.8). We firstly assume (without loss of generality) that the P_t^*s and w_ts are ordered so that $P_1^* \leq P_2^* \leq \cdots \leq P_T^*$. In this case, to minimize (2.8) over choice of the w_ts, we should choose a value $l \in \{1, \ldots, T\}$ and should set $w_t \leq \theta$ for $t = 1, \ldots, l$ and $w_t > \theta$ for $t = l+1, \ldots, T$. Since $\sum_{t=1}^{T} w_t = 1$ and $w_t > 0$, for each t such that $w_t \leq \theta$ we should set $w_t = \gamma$, where γ is some constant arbitrarily close to zero. So, for the first l leaves we should set $w_t = \gamma$. We should then set $w_t = (1 - l\gamma)/(T - l)$ for the remaining $T - l$ leaves. Since $w_t > \theta$ for these leaves, we should set $\theta = (1 - l\gamma)/(T - l) - \gamma$. For these choices, (2.8) reduces to being arbitrarily

close to

$$\mathbf{P}_D(\text{sign}(F(x)) \neq y) \leq \mathbf{P}_S(\text{sign}(F(x)) \neq y) + \sum_{t=1}^{l} P_t^* + (T-l)\sqrt{\frac{B}{m}}, \tag{2.9}$$

We have thus reduced the problem of choosing the w_ts and θ to minimize (2.8) to the problem of choosing a leaf index l to minimize (2.9). By making assumptions on the form of the cumulative correct leaf weight function, $\sum_{t=1}^{l} P_t^*$, we can find closed form solutions for this new minimization problem.

For instance, if we assume that $\sum_{t=1}^{l} P_t^*$ is bounded above by the function $c/(T-l)^2$, for some constant c, then we obtain the following result by optimizing over the choice of l. In the next section, we will see that this not an unreasonable assumption and can often be satisfied in practice.

Theorem 2.5

There exists a constant κ such that with probability at least $1-\delta$ over the random choice of S according to D^m, every decision tree whose cumulative correct leaf weight function, $\sum_{t=1}^{l} P_t^*$, is bounded above by $c/(T-l)^2$, for some constant c, satisfies

$$\begin{aligned}\mathbf{P}_D(\text{sign}(F(x)) \neq y) \quad &\leq \quad \mathbf{P}_S(\text{sign}(F(x)) \neq y) \\ &+\kappa\left(\frac{c}{m}\left((\nu \ln m + \ln d)\ln\left(m\theta^2/\nu\right) + \ln(1/\delta)\right)\right)^{1/3}\end{aligned}$$

where $\nu = \bar{d}\,\text{VCdim}(\mathcal{U})$.

Proof

Since $\sum_{t=1}^{l} P_t^*$ is bounded above by $c/(T-l)^2$, then for any $l \in \{0, \ldots, T-1\}$, Equation (2.9) is bounded above by

$$\mathbf{P}_D(\text{sign}(F(x)) \neq y) \leq \mathbf{P}_S(\text{sign}(F(x)) \neq y) + \frac{2.25c}{z^2} + z\sqrt{\frac{4B}{m}}, \tag{2.10}$$

where $z \in [T-l-\frac{1}{2}, T-l+\frac{1}{2}]$. We can now optimize over choice of the continuous parameter z by simply differentiating (2.10) with respect to z and equating to zero. This gives an optimal upper bound for (2.9) of

$$\mathbf{P}_D(\text{sign}(F(x)) \neq y) \leq \mathbf{P}_S(\text{sign}(F(x)) \neq y) + \left(\frac{18cB}{m}\right)^{1/3}. \tag{2.11}$$

Substitution for B then gives the required result. □

As before, provided the decision tree is not too asymmetric, $\bar{d}$ is roughly $\ln T$ and then

the result in Theorem 2.5 is roughly (ignoring constants and log terms),

$$\mathbf{P}_D(F(x) \neq y) \leq \mathbf{P}_S(F(x) \neq y) + \left(\frac{\ln T \, \mathrm{VCdim}(\mathcal{U})}{m}\right)^{1/3}.$$

Comparing this result to the VC bound in (2.7) we see that although our result will always be worse asymptotically (the rate in m is $-1/3$ compared to $-1/2$), it can be still be considerably smaller than the VC result for small m.

If we make stronger assumptions on the form of the cumulative correct leaf weight function (like it being exponential in l), then we can obtain stronger upper bounds on the generalization error with rates in m closer to the $-1/2$ of the VC result. However, our experimental results in the next section suggest that stronger assumptions may be difficult to guarantee in practice (at least when using traditional decision tree algorithms like C4.5).

2.4 Experiments and results

To examine the practical form of the cumulative correct leaf weight function, $\sum_{t=1}^{l} P_t^*$, we ran experiments on a collection of 10 data sets taken from the UC Irvine repository [11]. We restricted our attention to binary classification problems only. The data sets examined are listed in Table 2.1.

Data Set	Examples	Features
ionosphere	351	34
vote1	435	16
credit	690	15
breast-cancer	699	9
pima-indians	768	8
hypothyroid	3163	25
sick-euthyroid	3163	25
splice	3190	60
kr-vs-kp	3198	36
dis	3772	29

Table 2.1: Summary of examined data sets.

To build decision trees for these data sets, we used Quinlan's C4.5 algorithm [54] with default options and pruning enabled. Since the leaf weights are a function of the training examples, and we were not interested in the generalization error, we used the entire data set for training purposes in each case.

For six of the data sets (`breast-cancer`, `sick-euthyroid`, `credit`, `hypothyroid`, `dis` and `vote1`), we obtained decision trees whose cumulative correct leaf weight functions were well-fitted by upper bounds of the form $c/(T-l)^2$. The distributions for these data

sets (and their corresponding upper bounds with constant c minimized) are illustrated in Figure 2.1. Notice that, for all of these data sets, the optimal constants for the upper bounds are quite small and the upper bound is close to the cumulative correct leaf weight function.

This is not the case for the remaining four data sets (`kr-vs-kp`, `splice`, `pima-indians` and `ionosphere`). The cumulative correct leaf weight functions for the decision trees

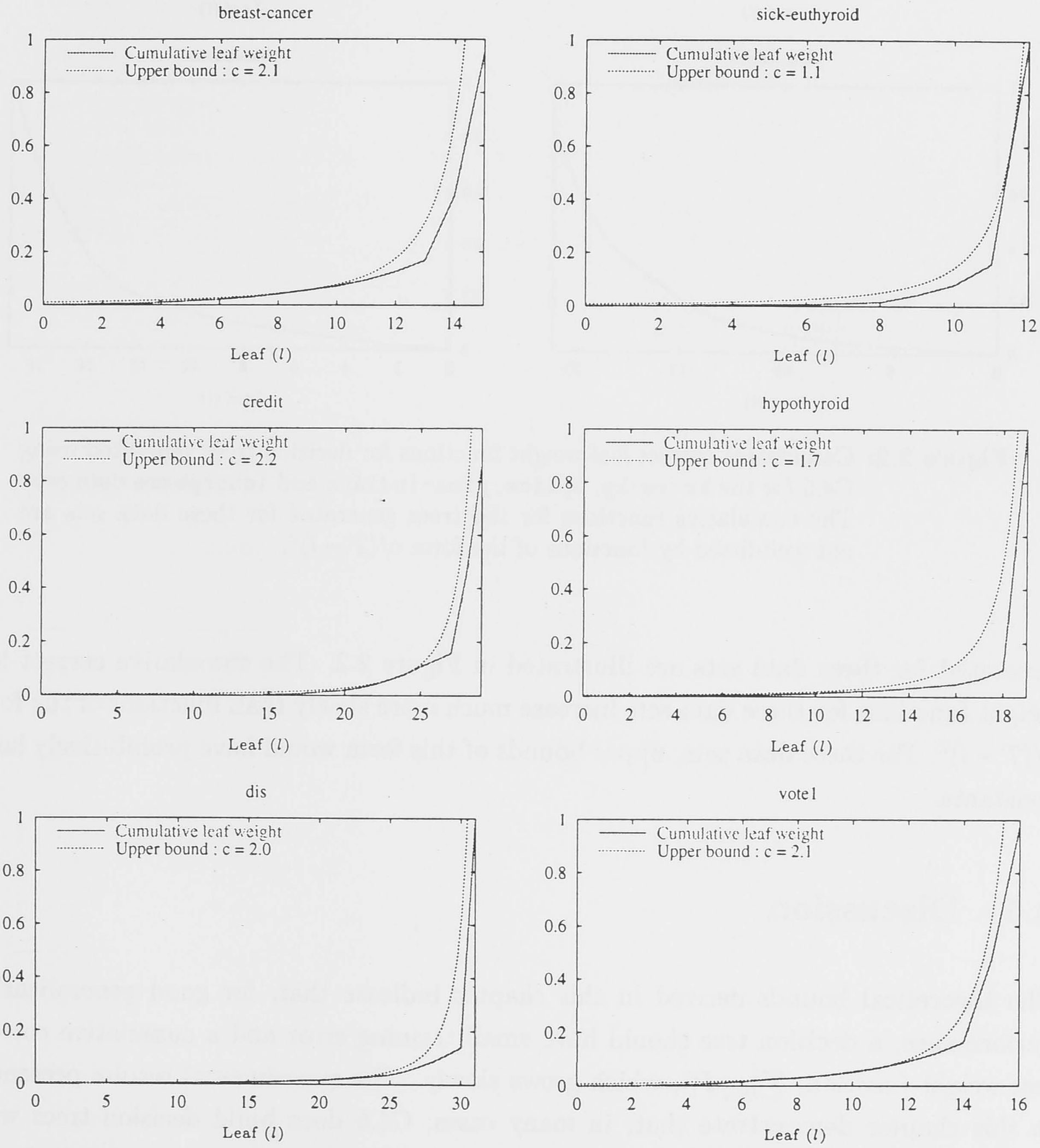


Figure 2.1: Cumulative correct leaf weight functions for decision trees generated using C4.5 for the `breast-cancer`, `sick-euthyroid`, `credit`, `hypothyroid`, `dis` and `vote1` data sets. Upper bounds on the cumulative functions of the form $c/(T-l)^2$ are also shown for comparison. The constant c is chosen in each case to minimize the upper bound.

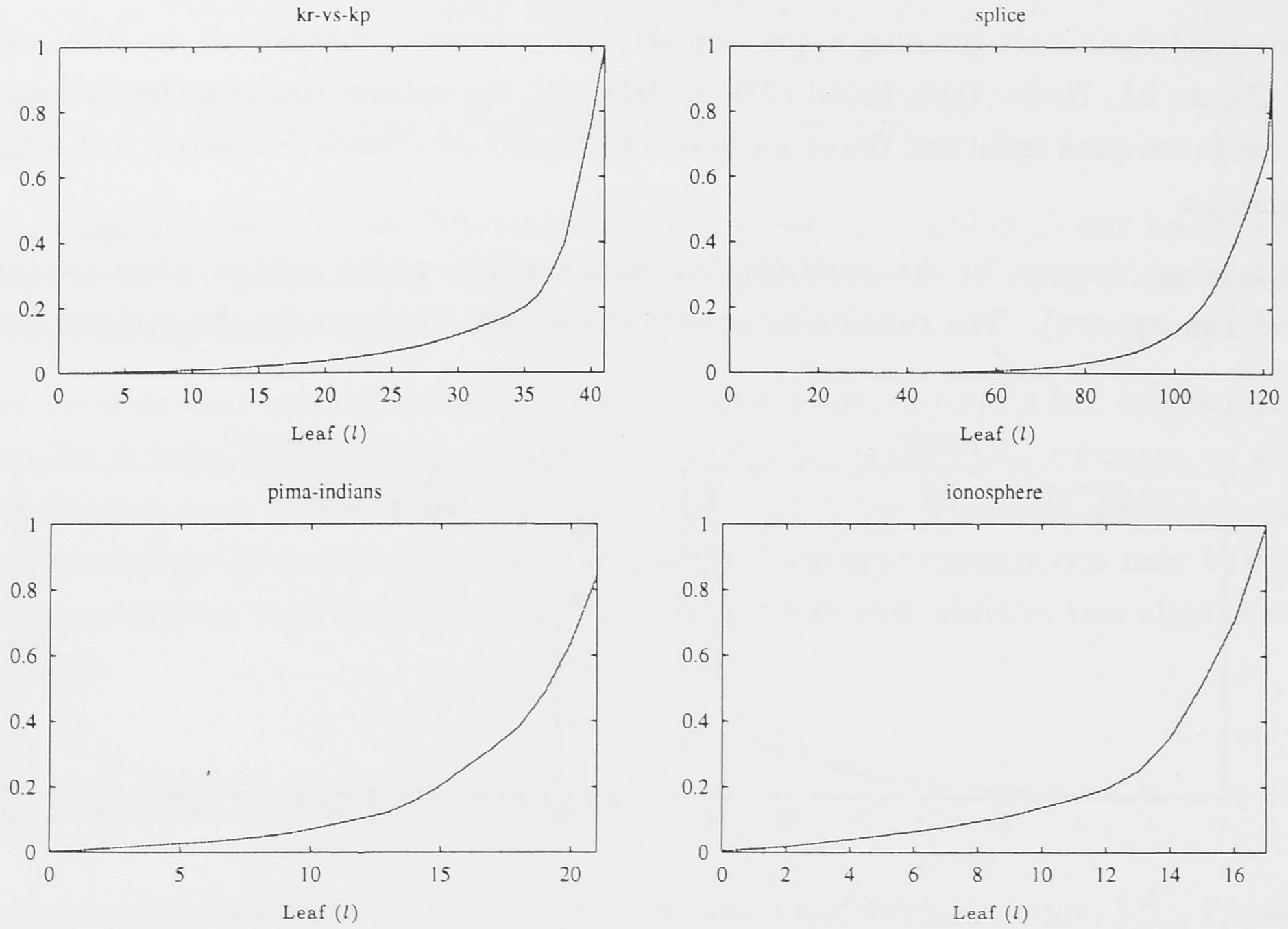


Figure 2.2: Cumulative correct leaf weight functions for decision trees generated using C4.5 for the `kr-vs-kp`, `splice`, `pima-indians` and `ionosphere` data sets. The cumulative functions for the trees generated for these data sets are not well-fitted by functions of the form $c/(T-l)^2$.

generated for these data sets are illustrated in Figure 2.2. The cumulative correct leaf weight functions for these data sets increase much more slowly than functions of the form $c/(T-l)^2$. For these data sets, upper bounds of this form would have prohibitively large constants.

2.5 Discussion

The theoretical bounds derived in this chapter indicate that, for good generalization performance, a decision tree should have small training error and a cumulative correct leaf weight function, $\sum_{t=1}^{l} P_t^*$, which grows slowly. The experimental results presented in this chapter demonstrate that, in many cases, C4.5 does build decision trees with cumulative correct leaf weight functions which satisfy this property. Future work should involve a more detailed experimental study of the relationship between training error, generalization error and the rate of growth of the cumulative correct leaf weight function. Following this, the goal would be to develop new algorithms which attempt to construct decision trees which balance the minimization of training error with the minimization of

the rate of growth of the cumulative correct leaf weight function.

CHAPTER

3 Threshold margin bounds for combinations of decision trees

■ ■ ■

In Chapter 2 we used existing techniques for the analysis of thresholded real-valued classifiers to bound the generalization error of a decision tree by representing the tree as a thresholded convex combination of leaf functions. The resulting bounds depended on the training error and on a complexity term related to the distribution of training examples over leaves.

In Section 3.1 of this chapter we use similar techniques to derive an upper bound on the generalization error of any thresholded convex combination of functions which are themselves thresholded convex combinations of functions in terms of the margin and the average complexity of the combined functions. The complexity of each of the combined functions in this result depends on the proportion of examples which are close to threshold.

Then, in Section 3.2, using the representation of a decision tree from Chapter 2, we apply our general result of Section 3.1 to derive an upper bound on the generalization error of any voted combination of decision trees in terms of the proportion of training examples with small margin and the average complexity of the decision trees (where the average is in terms of the voting weights). This improves on Theorem 1.9 for the special case of voted decision trees in two key ways. Firstly, the complexity of a single tree in the combination depends on a quantity which can be significantly smaller than the VC dimension of the class of decision trees. Secondly, instead of depending on the complexity of the most complex decision tree in the combination, the bound depends on the average complexity of all of the combined decision trees. Although this result can be considered to be, in some ways, a combination of Theorems 1.9 and 2.4, the proof is considerably more involved. Proofs are presented in Section 3.3.

3.1 Combinations of combinations

Within this chapter we restrict our attention to binary classifiers of the form sign $\big(F(x)\big)$ where

$$F(x) = \sum_{t=1}^{T} w_t \operatorname{sign}\big(f_t(x)\big) \tag{3.1}$$

and

$$f_t(x) = \sum_{s=1}^{T_t} v_{ts} e_{ts}(x), \tag{3.2}$$

with $\sum_{t=1}^{T} w_t = 1$, $w_t \geq 0$, $\sum_{s=1}^{T_t} v_{ts} = 1$, $v_{ts} \geq 0$ and $e_{ts} \in \mathcal{E}_{n_{ts}}$. Here $n_{ts} \in \{1, \ldots, r\}$ and $\mathcal{E}_1, \ldots, \mathcal{E}_r$ are classes of real-valued functions that map from X to $[-b, b]$ for some $b \in \mathbb{R}$. Expressing functions in this way allows us to calculate the complexity of the class of intermediate functions f_t in a refined manner. Instead of dealing with a single complex class of these functions, we can choose to consider several simpler classes. We denote the class of functions F of the form defined above by $\mathcal{F}$. Recall that, for a classifier which is a thresholded function from $\mathcal{F}$, the margin of an example (x, y) is $yF(x)$.

The following theorem bounds the generalization error of any function in $\mathcal{F}$ in terms of the proportion of training examples with small margin (less than some θ_0) and the average of the complexity of the functions f_t. The complexity of each function f_t is related to the proportion of training examples near threshold (within some θ_t of threshold).

Theorem 3.1
There exists a constant κ such that with probability at least $1 - \delta$ over random choice of the sample set S according to D^m, every $F \in \mathcal{F}$, $\theta_0 \in (0, 1]$ and $\theta_t \in (0, 1]$ satisfies

$$\begin{aligned} \mathbf{P}_D(yF(x) \leq 0) \quad \leq \quad & 2\mathbf{P}_S(yF(x) \leq \theta_0) \\ & + \kappa \frac{1}{m} \left[\frac{\ln m}{\theta_0^2} \left(\sum_{t=1}^{T} w_t \min(C_1, C_2) + C_3 \right) + \ln(1/\delta) \right], \end{aligned}$$

where

$$\begin{aligned} C_1 &= \frac{b^2 \ln(m/\theta_0)}{\theta_t^2} \left(\ln(rT_t) + \sum_{s=1}^{T_t} v_{ts} \ln \Pi_{\mathcal{E}_{n_{ts}}}(2m) \right) + \frac{m\theta_0}{\ln m} \mathbf{P}_S\big(|f_t(x)| < \theta_t\big), \\ C_2 &= T_t \ln(rm) + \sum_{s=1}^{T_t} \ln \Pi_{\mathcal{E}_{n_{ts}}}(2m), \\ C_3 &= \ln\left(\frac{Tb^2 \ln(m/\theta_0)}{\min_t \theta_t^2} \right). \end{aligned}$$

Due to its involved nature, the proof of this result is deferred to Section 3.3.1. The general idea is to use two random approximating functions, one approximating F by a random combination of f_t and one approximating the first approximation by approximating each of the f_t by a random combination of e_{ts}. Using these random approximations we can bound the probability of large generalization error by a probability depending only on the second approximation, which we then bound using a standard structural risk minimization result.

Theorem 3.1 provides a bound on generalization error which depends on two terms. The first term depends on $\mathbf{P}_S(yF(x) \leq \theta_0)$, the proportion of training examples with margin less than some threshold θ_0 (which we are free to choose). The second term depends on the average complexity of each f_t in the combination (where the average is in terms of the weight w_t assigned to each f_t) divided by θ_0^2 (which means we should choose θ_0 as large as possible). So to minimize the bound over choice of θ_0 we want to choose θ_0 as large as possible while ensuring that $\mathbf{P}_S(yF(x) \leq \theta_0)$ is as small as possible (i.e., as close as possible to the training error $\mathbf{P}_S(yF(x) \leq 0)$). If most training examples are classified with a large margin, then we can choose a large value of θ_0 and $\mathbf{P}_S(yF(x) \leq \theta_0)$ will still be close to $\mathbf{P}_S(yF(x) \leq 0)$. So, provided the classifier attains large margins on the training examples, we can optimize over θ_0 to give a reasonable result.

Turning now to the complexity of each f_t, we see that it is essentially the minimum of two alternative measures of complexity, C_1 and C_2. Dealing with the simpler case first, the value of C_2 is roughly proportional to the sum of the complexities of the functions e_{ts} which make up f_t — the complexities of these functions are measured by the growth function. The complexity measure C_1 depends on two terms. One term depends on $\mathbf{P}_S\big(|f_t(x)| < \theta_t\big)$, the proportion of training examples within some distance θ_t (which we are free to choose) of the threshold of f_t. The other term is roughly proportional to the average of the complexities of the functions e_{ts} divided by θ_t^2 (which means we should choose θ_t as large as possible). So to minimize the value of C_1 over choice of θ_t we want to choose θ_t as large as possible while ensuring that $\mathbf{P}_S\big(|f_t(x)| < \theta_t\big)$ is as close to zero as possible. If the output of f_t is far from threshold for most training examples, then we can choose a large value of θ_t and $\mathbf{P}_S\big(|f_t(x)| < \theta_t\big)$ will still be small. In this case, C_1 can be considerably smaller than C_2. On the hand, if the output of f_t is close to threshold on many training examples then, irrespective of the choice of θ_t, C_2 will most likely be smaller than C_1. Of course, since the bound depends on the average complexity of the f_t, the values of C_1 and C_2 are only relevant for those f_t which have significant weight in the combination.

So to obtain good bounds over choice of θ_0 and the θ_t we need our classifier to satisfy two conditions. Firstly, the classifier must classify most examples with a large margin. Secondly, the output of most of the functions f_t with large weight in the combination should be far from threshold on most training examples. In the following section we

shall see that when these two conditions are satisfied this bound can be applied to give significant improvements on existing results for voted combinations of decision trees.

A similar result to Theorem 3.1 can be obtained with $2\mathbf{P}_S(yF(x) \leq \theta_0)$ replaced by $\mathbf{P}_S(yF(x) \leq \theta_0)$ at the expense of an exponent of one half on the term decreasing with m. Also, considering that Theorems 1.9 and 2.4 for general convex combinations of classifiers have no dependence on the number of classifiers T in the combination, it seems likely that the logarithmic dependence on T in this result is an artifact of the analysis.

3.2 Combinations of decision trees

In this section, we apply the theorem obtained in Section 3.1 to the problem of bounding the generalization error of a convex combination of decision trees. We first represent a decision tree as a thresholded convex combination of Boolean functions. As in Chapter 2, this can be accomplished in the following manner.

Given a decision tree with T_t leaves, let $\sigma_{ts} \in \{\pm 1\}$ denote the label associated with leaf s and $L_{ts}(x)$ be the $\{0,1\}$–valued function producing 1 if and only if x reaches leaf s. Then for any sequence of v_{ts} such that $v_{ts} > 0$ and $\sum_{s=1}^{T_t} v_{ts} = 1$ the output of the tree for an example x can be written as

$$\text{sign}\left(\sum_{s=1}^{T_t} v_{ts}\sigma_{ts}L_{ts}(x)\right).$$

Now setting $e_{ts}(x) = \sigma_{ts}L_{ts}(x)$ gives that the output of the tree for an example x can be written as $\text{sign}(f_t(x))$ where $f_t(x)$ is defined as in (3.2).

We now define, for each tree t, a probability distribution $P_t = \{P_{ts}\}$ over leaves via $P_{ts} = \mathbf{P}_S(L_{ts}(x) = 1)$. Thus P_{ts} is simply the proportion of training examples which are classified using leaf s of tree t.

Denoting the depth of leaf s in tree t by d_{ts} and the maximum depth of leaves in tree t by d_t we define the average depth $\bar{d}_t$ of tree t relative to leaf weights v_{ts} by

$$\bar{d}_t = \sum_{s=1}^{T_t} v_{ts}d_{ts}.$$

The following theorem extends the main result of Chapter 2 to handle the case where multiple decision trees are combined as a thresholded convex combination. In this case the bound is in terms of the proportion of examples with small margin and the average complexity of the individual decision trees.

Theorem 3.2

There exists a constant κ such that with probability at least $1-\delta$ over random choice of the sample set S according to D^m, every convex combination F of decision trees, $\theta_0 \in (0,1]$ and $\theta_t \in (0,1]$ satisfies

$$\begin{aligned}\mathbf{P}_D(yF(x) \leq 0) \quad \leq \quad & 2\mathbf{P}_S(yF(x) \leq \theta_0) \\ & +\kappa\frac{1}{m}\left[\frac{\ln m}{\theta_0^2}\left(\sum_{t=1}^{T} w_t \min(C_1, C_2) + C_3\right) + \ln(1/\delta)\right],\end{aligned}$$

where

$$\begin{aligned}C_1 &= \frac{\ln(m/\theta_0)}{\theta_t^2}\left(\bar{d}_t \,\mathrm{VCdim}(\mathcal{U}) \ln m + \ln(T_t \max_l d_l)\right) + \frac{m\theta_0}{\ln m}\sum_{s=1}^{T_t} P_{ts} 1(v_{ts} < \theta_t), \\ C_2 &= T_t\Big(\mathrm{VCdim}(\mathcal{U}) \ln m + \ln(\max_l d_l)\Big), \\ C_3 &= \ln\left(\frac{T\ln(m/\theta_0)}{\min_t \theta_t^2}\right).\end{aligned}$$

where $\mathcal{U}$ denotes the class of node decision functions and $1(v_{ts} < \theta_t)$ is 1 if $v_{ts} < \theta_t$ and 0 otherwise.

The proof of this result is presented in Section 3.3.2. To understand the significance of this result let us first consider existing results for voted combinations of decision trees. After bounding the VC dimension of the class of convex combinations of decision trees (see, for example, Chapter 21 in [20]) we can apply a standard VC result (using a comparable rate to that of Theorem 3.2) to show that roughly (ignoring constants and log terms)

$$\mathbf{P}_D(yF(x) \leq 0) \leq 2\mathbf{P}_S(yF(x) \leq 0) + \frac{\mathrm{VCdim}(\mathcal{U})}{m} T \max_t T_t.$$

By applying Theorem 2.2 for the specific case of combined decision trees we can obtain a result which is roughly (again ignoring constants and log terms)

$$\mathbf{P}_D(yF(x) \leq 0) \leq 2\mathbf{P}_S(yF(x) \leq \theta_0) + \frac{\mathrm{VCdim}(\mathcal{U})}{m\theta_0^2}\sum_{t=1}^{T} w_t T_t. \tag{3.3}$$

Note that Theorem 1.9 implies a similar result with $\sum_{t=1}^{T} w_t T_t$ replaced by $\max_t T_t$.

In the bound in (3.3) we want to choose θ_0 as large as possible (to minimize the second term on the right hand side) while ensuring that $\mathbf{P}_S(yF(x) \leq \theta_0)$ is not much larger than $\mathbf{P}_S(yF(x) \leq 0)$. If we choose θ_0 in this way then we are comparing $T\max_t T_t$ in the VC bound to $\sum_{t=1}^{T} w_t T_t/\theta_0^2$ in the bound in (3.3). If the combined classifier attains large margins on the training examples then it will be possible to choose a large value of θ_0 and in this case the bound in (3.3) will be significantly smaller than the VC bound.

Returning to Theorem 3.2, let us consider the case when $C_1 \leq C_2$ for all t. In this case the result is roughly (again ignoring constants and log terms)

$$\mathbf{P}_D(yF(x) \leq 0) \leq 2\mathbf{P}_S(yF(x) \leq \theta_0) + \frac{1}{\theta_0}\sum_{t=1}^{T} w_t \sum_{s=1}^{T_t} P_{ts} 1(v_{ts} < \theta_t) + \frac{\mathrm{VCdim}(\mathcal{U})}{m\theta_0^2}\sum_{t=1}^{T} w_t \frac{\bar{d}_t + \ln T_t}{\theta_t^2}.$$

To minimize this bound, we want to choose θ_0 in exactly the same way as for the bound in (3.3). However, we also want to choose the θ_t as large as possible (to minimize the third term on the right hand side) while ensuring that $\sum_{s=1}^{T_t} P_{ts} 1(v_{ts} < \theta_t)$ is close to zero. If we assume that the decision trees are not too asymmetric, the depth of a tree is roughly the logarithm of the number of leaves. In this case, we are comparing $\sum_{t=1}^{T} w_t T_t$ in the bound in (3.3) to $\sum_{t=1}^{T} w_t \ln T_t/\theta_t^2$ in our result. If each decision tree in the combination uses a small number of leaves to classify most training examples (i.e., the distribution $\{P_{ts}\}$ is skewed) then it will be possible to choose large values of θ_t (as $\sum_{s=1}^{T_t} P_{ts} 1(v_{ts} < \theta_t)$ will be small even for large θ_t) and our result will be significantly smaller than the bound in (3.3). Any trees in the combination which do not satisfy this assumption will have $C_2 < C_1$ (because we cannot choose θ_t large enough) and in this case the complexity is roughly T_t and we do no worse for these trees than the bound in (3.3).

3.3 Proofs

3.3.1 Proof of Theorem 3.1

Proof

In what follows, all random choices are independent. We define a random approximating function for $F \in \mathcal{F}$ using

$$G(x) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{sign}\left(f_{l_i}(x)\right)$$

where the l_i are chosen randomly with $\mathbf{P}(l_i = z) = w_z$. The class of such functions for a particular F and N will be denoted by $\mathcal{G}$. From this we define a random approximating function for $G \in \mathcal{G}$ using

$$H(x) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{sign}\left(h_i(x)\right)$$

where

$$h_i(x) = \begin{cases} \dfrac{1}{N_{l_i}}\displaystyle\sum_{j=1}^{N_{l_i}} e_{l_i m_{ij}}(x) & : \text{ if } \lambda_{l_i} = 1 \\ f_{l_i}(x) & : \text{ if } \lambda_{l_i} = 0 \end{cases}$$

where the m_{ij} are chosen randomly with $\mathbf{P}(m_{ij} = z) = v_{l_i z}$ and $\lambda_{l_i} \in \{0, 1\}$. The λ_t allow us to specify which of the f_t are approximated randomly. The class of such functions for a particular $F, N, l_1, \ldots, l_N, N_{l_1}, \ldots, N_{l_N}$ and $\lambda_{l_1}, \ldots, \lambda_{l_N}$ will be denoted by $\mathcal{H}$. The probability of choosing a $G \in \mathcal{G}$ randomly which satisfies a predicate A will be denoted by $\mathbf{P}_G(A)$ and similarly $\mathbf{P}_H(A)$ will denote the probability of choosing a random $H \in \mathcal{H}$ satisfying A. Similarly, $\mathbf{E}_G$ and $\mathbf{E}_H$ will denote expectation with respect to random choice of G and H respectively. In order to simplify what follows we define the following indicator functions. Here, $1(A)$ denotes the indicator function of a predicate A.

$$
\begin{aligned}
R &= 1\Big(yF(x) \leq 4\theta_0 \vee \sum_{t=1}^{T} \{w_t : |f_t(x)| < 5\theta_t \lambda_t\} > \theta_0/2\Big), \\
\Upsilon_1 &= 1\Big(yG(x) \leq \theta_0 \vee \tfrac{1}{N} |\{i : |f_{l_i}(x)| < \theta_{l_i} \lambda_{l_i}\}| > \theta_0\Big), \\
\Phi_1 &= 1\Big(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N} |\{i : |h_i(x)| < 2\theta_{l_i} \lambda_{l_i}\}| > \theta_0\Big), \\
\Phi_2 &= 1\Big(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N} |\{i : |h_i(x)| < 4\theta_{l_i} \lambda_{l_i}\}| > \theta_0\Big), \\
\Upsilon_2 &= 1\Big(yG(x) \leq 3\theta_0 \vee \tfrac{1}{N} |\{i : |f_{l_i}(x)| < 5\theta_{l_i} \lambda_{l_i}\}| > \theta_0\Big).
\end{aligned}
$$

To prove the result we require the following lemma which allows us to relate the probability of the event in the theorem to a standard structural risk minimization result.

Lemma 3.3

For all $F \in \mathcal{F}$, $N > 0$, $G \in \mathcal{G}$, $N_t > 0$, $x \in X$, $y \in \{\pm 1\}$, $\lambda_t \in \{0, 1\}$, $\theta_0 \in (0, 1]$ *and* $\theta_t \in (0, 1]$

$$
\begin{aligned}
&(a) \quad \mathbf{E}_H \Phi_1 \geq \Upsilon_1 - 2 \textstyle\sum_{i=1}^{N} \lambda_{l_i} e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2} \\
&(b) \quad \mathbf{E}_H \Phi_2 \leq \Upsilon_2 + 4 \textstyle\sum_{i=1}^{N} \lambda_{l_i} e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}
\end{aligned}
$$

For all $F \in \mathcal{F}$, $N > 0$, $x \in X$, $y \in \{\pm 1\}$ *and* $\theta_0 \in (0, 1]$

$$
\begin{aligned}
&(c) \quad \mathbf{E}_G \Upsilon_1 \geq 1(yF(x) \leq 0) - e^{-{\theta_0}^2 N/2} \\
&(d) \quad \mathbf{E}_G \Upsilon_2 \leq R + 2e^{-{\theta_0}^2 N/2}
\end{aligned}
$$

Proof

(a) We need to show that

$$
\begin{aligned}
&\mathbf{P}_H \left(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N} |\{i : |h_i(x)| < 2\theta_{l_i} \lambda_{l_i}\}| > \theta_0\right) \\
&\qquad \geq 1\left(yG(x) \leq \theta_0 \vee \tfrac{1}{N} |\{i : |f_{l_i}(x)| < \theta_{l_i} \lambda_{l_i}\}| > \theta_0\right) - 2 \sum_{i=1}^{N} \lambda_{l_i} e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}.
\end{aligned}
$$

Clearly the inequality holds if the first term on the right hand side is zero. We consider separately the two cases when it is one. If $yG(x) \leq \theta_0$ and $\frac{1}{N}|\{i : |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}\}| \leq \theta_0$ we then need

$$1 - \mathbf{P}_H\left(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N}|\{i : |h_i(x)| < 2\theta_{l_i}\lambda_{l_i}\}| > \theta_0\right) \leq 2\sum_{i=1}^{N} \lambda_{l_i} e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}. \quad (3.4)$$

Now

$$1 - \mathbf{P}_H(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N}|\{i : |H_i(x)| < 2\theta_{l_i}\lambda_{l_i}\}| > \theta_0) \leq \mathbf{P}_H(yH(x) > 2\theta_0)$$

and

$$\mathbf{P}_H(yH(x) > 2\theta_0) \leq \mathbf{P}_H(yH(x) - yG(x) > \theta_0)$$

as $yG(x) \leq \theta_0$. Hence

$$\begin{aligned}
\mathbf{P}_H(yH(x) > 2\theta_0) &\leq \mathbf{P}_H\Big(\tfrac{1}{N}\big|\{i : \operatorname{sign}(yh_i(x)) > \operatorname{sign}(yf_{l_i}(x))\}\big| > \theta_0\Big) \\
&\leq \mathbf{P}_H\big(\exists i : \operatorname{sign}(yh_i(x)) = 1 \wedge yf_{l_i}(x) \leq -\theta_{l_i}\lambda_{l_i}\big) \\
&\qquad \big[\text{since } \tfrac{1}{N}|\{i : |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}\}| \leq \theta_0\big] \\
&\leq \sum_{i=1}^{N} \lambda_{l_i} \mathbf{P}_H(\operatorname{sign}(yh_i(x)) = 1 \wedge yf_{l_i}(x) \leq -\theta_{l_i}\lambda_{l_i}) \\
&\qquad \big[\text{by applying the union bound}\big] \\
&\leq \sum_{i=1}^{N} \lambda_{l_i} \mathbf{P}_H(y(h_i(x) - f_{l_i}(x)) \geq \theta_{l_i}\lambda_{l_i}) \\
&\leq \sum_{i=1}^{N} \lambda_{l_i} e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}
\end{aligned}$$

by applying Hoeffding's inequality. If $\frac{1}{N}|\{i : |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}\}| > \theta_0$ we again need (3.4) to hold. Now

$$\begin{aligned}
&1 - \mathbf{P}_H(yH(x) \leq 2\theta_0 \vee \tfrac{1}{N}|\{i : |h_i(x)| < 2\theta_{l_i}\lambda_{l_i}\}| > \theta_0) \\
&\qquad \leq \mathbf{P}_H(\tfrac{1}{N}|\{i : |h_i(x)| < 2\theta_{l_i}\lambda_{l_i}\}| \leq \theta_0)
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{P}_H(\tfrac{1}{N}|\{i : |h_i(x)| < 2\theta_{l_i}\lambda_{l_i}\}| \leq \theta_0) &\leq \mathbf{P}_H(\exists i : |h_i(x)| \geq 2\theta_{l_i}\lambda_{l_i} \wedge |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}) \\
&\qquad \big[\text{as } \tfrac{1}{N}|\{i : |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}\}| > \theta_0\big] \\
&\leq \mathbf{P}_H(\exists i : |h_i(x)| - |f_{l_i}(x)| > \theta_{l_i}\lambda_{l_i}) \\
&\leq \mathbf{P}_H(\exists i : |h_i(x) - f_{l_i}(x)| > \theta_{l_i}\lambda_{l_i})
\end{aligned}$$

$$\leq 2\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}$$

by applying the union bound and Hoeffding's inequality.

(b) We need to show that

$$\mathbf{P}_H\left(yH(x)\leq 2\theta_0 \vee \tfrac{1}{N}|\{i:|h_i(x)|<4\theta_{l_i}\lambda_{l_i}\}|>\theta_0\right)$$
$$\leq 1\left(yG(x)\leq 3\theta_0 \vee \tfrac{1}{N}|\{i:|f_{l_i}(x)|<5\theta_{l_i}\lambda_{l_i}\}|>\theta_0\right)+4\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}.$$

We only need to consider the case when the first term on the right hand side is zero, since the bound is trivial otherwise. If $yG(x)>3\theta_0$ and $\frac{1}{N}|\{i:|f_{l_i}(x)|<5\theta_{l_i}\lambda_{l_i}\}|\leq\theta_0$ we then need

$$\mathbf{P}_H(yH(x)\leq 2\theta_0 \vee \tfrac{1}{N}|\{i:|h_i(x)|<4\theta_{l_i}\lambda_{l_i}\}|>\theta_0)\leq 4\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}.$$

In order to bound this probability we apply the union bound and bound the two resulting probabilities separately. Since $yG(x)>3\theta_0$, if $yH(x)\leq 2\theta_0$ then $yG(x)-yH(x)>\theta_0$, so

$$\mathbf{P}_H(yH(x)\leq 2\theta_0)\leq \mathbf{P}_H(yG(x)-yH(x)>\theta_0)$$
$$\leq\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}$$

by following the same argument as in the first case of part (a) using the fact that $\frac{1}{N}|\{i:|f_{l_i}(x)|<5\theta_{l_i}\lambda_{l_i}\}|\leq\theta_0$ and thus $\frac{1}{N}|\{i:|f_{l_i}(x)|<\theta_{l_i}\lambda_{l_i}\}|\leq\theta_0$. The second probability is

$$\mathbf{P}_H(\tfrac{1}{N}|\{i:|h_i(x)|<4\theta_{l_i}\lambda_{l_i}\}|>\theta_0)\leq \mathbf{P}_H(\exists i:|h_i(x)|<4\theta_{l_i}\lambda_{l_i}\wedge|f_{l_i}(x)|\geq 5\theta_{l_i}\lambda_{l_i})$$
$$\left[\text{as } \tfrac{1}{N}|\{i:|f_{l_i}(x)|<5\theta_{l_i}\lambda_{l_i}\}|\leq\theta_0\right]$$
$$\leq \mathbf{P}_H(\exists i:|f_{l_i}(x)|-|h_i(x)|>\theta_{l_i}\lambda_{l_i})$$
$$\leq 2\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}$$

by the same argument as that in the second case of part (a).

(c) We need to show that

$$\mathbf{P}_G\left(yG(x)\leq\theta_0\vee\tfrac{1}{N}|\{i:|f_{l_i}(x)|<\theta_{l_i}\lambda_{l_i}\}|>\theta_0\right)\geq 1\left(yF(x)\leq 0\right)-e^{-{\theta_0}^2N/2}.$$

If $yF(x)\leq 0$ we then need

$$1-\mathbf{P}_G(yG(x)\leq\theta_0\vee\tfrac{1}{N}|\{i:|f_{l_i}(x)|<\theta_{l_i}\lambda_{l_i}\}|>\theta_0)\leq e^{-{\theta_0}^2N/2}.$$

Now

$$1 - \mathbf{P}_G(yG(x) \leq \theta_0 \vee \tfrac{1}{N}|\{i : |f_{l_i}(x)| < \theta_{l_i}\lambda_{l_i}\}| > \theta_0) \leq \mathbf{P}_G(yG(x) > \theta_0)$$

and $\mathbf{P}_G(yG(x) > \theta_0) \leq \mathbf{P}_G(yG(x) - yF(x) > \theta_0)$ as $yF(x) \leq 0$. Hoeffding's inequality implies that

$$\mathbf{P}_G(yG(x) - yF(x) > \theta_0) \leq e^{-{\theta_0}^2 N/2}.$$

(d) We need to show that

$$\begin{aligned}\mathbf{P}_G\left(yG(x) \leq 3\theta_0 \vee \tfrac{1}{N}|\{i : |f_{l_i}(x)| < 5\theta_{l_i}\lambda_{l_i}\}| > \theta_0\right)\\ \leq 1\left(yF(x) \leq 4\theta_0 \vee \sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} > \theta_0/2\right) + 2e^{-{\theta_0}^2 N/2}.\end{aligned}$$

If $yF(x) > 4\theta_0$ and $\sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} \leq \theta_0/2$ we then need

$$P_G(yG(x) \leq 3\theta_0 \vee \tfrac{1}{N}|\{i : |f_{l_i}(x)| < 5\theta_{l_i}\lambda_{l_i}\}| > \theta_0) \leq 2e^{-{\theta_0}^2 N/2}.$$

Once again we bound this probability by applying the union bound and bounding the two resulting probabilities separately. The first probability is bounded above by $\mathbf{P}_G(yF(x) - yG(x) > \theta_0)$ as $yF(x) > 4\theta_0$. Hoeffding's inequality implies that

$$\mathbf{P}_G(yF(x) - yG(x) > \theta_0) \leq e^{-{\theta_0}^2 N/2}.$$

The second probability is

$$\begin{aligned}P_G(\tfrac{1}{N}|\{i : |f_{l_i}(x)| < 5\theta_{l_i}\lambda_{l_i}\}| > \theta_0)\\ \leq \mathbf{P}_G\left(\tfrac{1}{N}|\{i : |f_{l_i}(x)| < 5\theta_{l_i}\lambda_{l_i}\}| > \sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} + \theta_0/2\right)\end{aligned}$$

since

$$\sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} \leq \theta_0/2.$$

Hoeffding's inequality shows that this probability is no more than $e^{-{\theta_0}^2 N/2}$. □

We are now going to bound $\mathbf{P}_D(yF(x) \leq 0)$ in terms of $\mathbf{E}_S(R)$ and a complexity term c (to be defined later), and then by suitable choice of the λ_t obtain the result.

$$\begin{aligned}&\mathbf{P}_{D^m}\Big(\exists F \in \mathcal{F}, N > 0, N^* > 0, 0 < N_t < N^*, \lambda_t \in \{0,1\}, \theta_0 \in (0,1], \theta_t \in (0,1] :\\ &\qquad \mathbf{P}_D(yF(x) \leq 0) > 2\mathbf{E}_S(R) + \mathbf{E}_G\mathbf{E}_H c + 5e^{-\theta_0^2 N/2} + \mathbf{E}_G 10\textstyle\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)\\ \leq\ &\mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, \lambda_t, \theta_0, \theta_t :\end{aligned}$$

$$\mathbf{E}_D\mathbf{E}_G\Upsilon_1 > 2\mathbf{E}_S R + \mathbf{E}_G\mathbf{E}_H c + 4e^{-\theta_0^2 N/2} + \mathbf{E}_G 10\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)$$

[by applying Lemma 3.3(c)]

$$\leq \mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, \lambda_t, \theta_0, \theta_t :$$

$$\mathbf{E}_D\mathbf{E}_G\Upsilon_1 > 2\mathbf{E}_S\mathbf{E}_G\Upsilon_2 + \mathbf{E}_G\mathbf{E}_H c + \mathbf{E}_G 10\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)$$

[by applying Lemma 3.3(d)]

$$\leq \mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, G \in \mathcal{G}, \lambda_t, \theta_0, \theta_t :$$

$$\mathbf{E}_D\Upsilon_1 > 2\mathbf{E}_S\Upsilon_2 + \mathbf{E}_H c + 10\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)$$

$$\leq \mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, G, \lambda_t, \theta_0, \theta_t :$$

$$\mathbf{E}_D\mathbf{E}_H\Phi_1 > 2\mathbf{E}_S\Upsilon_2 + \mathbf{E}_H c + 8\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)$$

[by applying Lemma 3.3(a)]

$$\leq \mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, G, \lambda_t, \theta_0, \theta_t : \mathbf{E}_D\mathbf{E}_H\Phi_1 > 2\mathbf{E}_S\mathbf{E}_H\Phi_2 + \mathbf{E}_H c\Big)$$

[by applying Lemma 3.3(b)]

$$\leq \mathbf{P}_{D^m}\Big(\exists F, N, N^*, N_t, G, \lambda_t, H \in \mathcal{H}, \theta_0, \theta_t : \mathbf{E}_D\Phi_1 > 2\mathbf{E}_S\Phi_2 + c\Big)$$

$$= \mathbf{P}_{D^m}\Big(\exists N, N^*, l_i, N_{l_i}, \lambda_{l_i}, m_{ij}, n_{l_i s}, H' \in \mathcal{H}', \theta_0, \theta_{l_i} : \mathbf{E}_D\Phi_1 > 2\mathbf{E}_S\Phi_2 + c\Big), \qquad (3.5)$$

where, for fixed $N, l_i, N_{l_i}, \lambda_{l_i}, m_{ij}, n_{l_i s}$, $\mathcal{H}'$ denotes the class of functions of the form

$$H'(x) = \frac{1}{N}\sum_{i=1}^{N}\operatorname{sign}(h'_i(x)),$$

where

$$h'_i(x) = \begin{cases} \dfrac{1}{N_{l_i}}\displaystyle\sum_{j=1}^{N_{l_i}} e'_{l_i m_{ij}}(x) & : \text{ if } \lambda_{l_i} = 1 \\ \displaystyle\sum_{s=1}^{T_{l_i}} v'_{l_i s}e'_{l_i s}(x) & : \text{ if } \lambda_{l_i} = 0, \end{cases} \qquad (3.6)$$

where $e'_{l_i m_{ij}} \in \mathcal{E}_{n_{l_i m_{ij}}}$ for $\lambda_{l_i} = 1$ and $e'_{l_i s} \in \mathcal{E}_{n_{l_i s}}$ for $\lambda_{l_i} = 0$. So the only thing varying over this class is the particular choices of the $e'_{l_i m_{ij}}$ and $e'_{l_i s}$ (the classes they are chosen from are fixed) and the $v'_{l_i s}$ which are restricted so that $v'_{l_i s} \in [0,1]$ and $\sum_{s=1}^{T_{l_i}} v'_{l_i s} = 1$.

The probability (3.5) can in turn be bounded above by

$$\sum_{N=1}^{\infty}\sum_{N^*=1}^{\infty} T^N {N^*}^N 2^N \max_{l_i, N_{l_i}, \lambda_{l_i}} \prod_{i=1}^{N}(rT_{l_i})^{\lambda_{l_i}N_{l_i}} r^{(1-\lambda_{l_i})T_{l_i}} 2(N+2) \max_{m_{ij}, n_{l_i s}, \theta_0} B, \qquad (3.7)$$

where

$$B = \mathbf{P}_{D^m}\big(\exists H' \in \mathcal{H}', \theta_{l_i} : \mathbf{E}_D\Phi_1 > 2\mathbf{E}_S\Phi_2 + c\big). \qquad (3.8)$$

Equation (3.7) follows from repeated application of the union bound over N, N^*, l_i, N_{l_i}, λ_{l_i}, m_{ij}, $n_{l_i s}$ and θ_0. The argument for the number of possible choices for each of these variables is as follows. There are N values of l_i and N_{l_i} to choose from $\{1,\ldots,T\}$ and $\{1,\ldots,N^*\}$ respectively. The λ_{l_i}s can be chosen in 2^N ways. For each i such that $\lambda_{l_i}=1$, the set of m_{ij}s can be chosen in $T_{l_i}^{N_{l_i}}$ ways and the set of $n_{l_i m_{ij}}$s can be chosen in $r^{N_{l_i}}$ ways. For each i such that $\lambda_{l_i}=0$, the set of $n_{l_i s}$s can be chosen in $r^{T_{l_i}}$ ways. The rest of the m_{ij}s and $n_{l_i s}$s make no difference. Since $H'\in\mathcal{H}'$ is the sum of N $\{\pm1\}$-valued variables there are only $2(N+2)$ values of θ_0 that make any difference in the definitions of Φ_1 and Φ_2.

In order to bound B and at the same time make the result uniform over choice of the θ_{l_i} we use a quantization trick similar to that of Proposition 8 in [5]. Let us now suppose that c is a non-increasing function $c(\theta_{l_1},\ldots,\theta_{l_N})$ of the θ_{l_i}. Our final choice of c will ensure that this is true. Note that $\mathbb{N}_0$ denotes the set of non-negative integers.

$$
\begin{aligned}
&\mathbf{P}_{D^m}\big(\exists H'\in\mathcal{H}',\theta_{l_i}\in(0,1]:\mathbf{E}_D\Phi_1>2\mathbf{E}_S\Phi_2+c\big)\\
&\quad=\ \mathbf{P}_{D^m}\Big(\exists H'\in\mathcal{H}',\theta_{l_i}\in(0,1]:\\
&\qquad\qquad\mathbf{E}_D\big(1\big(yH'(x)\le 2\theta_0\vee\tfrac{1}{N}|\{i:|h_i'(x)|<2\theta_{l_i}\lambda_{l_i}\}|>\theta_0\big)\big)\\
&\qquad\qquad>2\mathbf{E}_S\big(1\big(yH'(x)\le 2\theta_0\vee\tfrac{1}{N}|\{i:|h_i'(x)|<4\theta_{l_i}\lambda_{l_i}\}|>\theta_0\big)\big)+c\Big)\\
&\quad=\ \mathbf{P}_{D^m}\Bigg(\exists H'\in\mathcal{H}':S\in\bigcup_{q=1}^{N}\bigcup_{j_q\in I_q}\\
&\qquad\qquad\Big\{\mathbf{E}_D\big(1\big(yH'(x)\le 2\theta_0\vee\tfrac{1}{N}|\{i:|h_i'(x)|<2\theta_{l_i}\lambda_{l_i}\}|>\theta_0\big)\big)\\
&\qquad\qquad>2\mathbf{E}_S\big(1\big(yH'(x)\le 2\theta_0\vee\tfrac{1}{N}|\{i:|h_i'(x)|<4\theta_{l_i}\lambda_{l_i}\}|>\theta_0\big)\big)\\
&\qquad\qquad+c(\theta_{l_1},\ldots,\theta_{l_N}):\theta_{l_i}\in\big(\tfrac{1}{2}^{j_i+1},\tfrac{1}{2}^{j_i}\big]\Big\}\Bigg),
\end{aligned}
$$

where

$$
I_q=\begin{cases}1 & \text{if } \lambda_{l_q}=1\\ \mathbb{N}_0 & \text{if } \lambda_{l_q}=0\end{cases}.
$$

Notice that the probability is independent of the value of θ_{l_i} when $\lambda_{l_i}=0$. Hence, the value of j_q is not important when $\lambda_{l_q}=0$. In such cases, we fix $j_q=1$. Using the fact that c is a non-increasing function, this last probability is no more than

$$
\begin{aligned}
&\mathbf{P}_{D^m}\Bigg(\exists H'\in\mathcal{H}':S\in\bigcup_{q=1}^{N}\bigcup_{j_q\in I_q}\\
&\qquad\Big\{\mathbf{E}_D\big(1\big(yH'(x)\le 2\theta_0\vee\tfrac{1}{N}|\{i:|h_i'(x)|<2\ \tfrac{1}{2}^{j_i}\lambda_{l_i}\}|>\theta_0\big)\big)
\end{aligned}
$$

$$> 2\mathbf{E}_S\Big(1\big(yH'(x) \le 2\theta_0 \vee \tfrac{1}{N}|\{i : |h'_i(x)| < 4\ \tfrac{1}{2}^{j_i+1}\lambda_{l_i}\}| > \theta_0\big)\Big)$$
$$+c\big(\tfrac{1}{2}^{j_1},\ldots,\tfrac{1}{2}^{j_N}\big)\Big\}\Big)$$

$$\le \sum_{q=1}^{N}\sum_{j_q\in I_q}\mathbf{P}_{D^m}\Big(\exists H' \in \mathcal{H}' :$$
$$\mathbf{E}_D\big(1\big(yH'(x) \le 2\theta_0 \vee \tfrac{1}{N}|\{i : |h'_i(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\big)\big)$$
$$> 2\mathbf{E}_S\big(1\big(yH'(x) \le 2\theta_0 \vee \tfrac{1}{N}|\{i : |h'_i(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\big)\big)$$
$$+c\big(\tfrac{1}{2}^{j_1},\ldots,\tfrac{1}{2}^{j_N}\big)\Big). \tag{3.9}$$

In order to bound this last probability we now apply Lemma 1.3 with

$$\Psi = \left\{1\left(yH'(x) \le 2\theta_0 \vee \tfrac{1}{N}|\{i : |h'_i(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\right) : H' \in \mathcal{H}'\right\}.$$

For this choice of Ψ, $\Pi_\Psi(m)$ is certainly no more than $\Pi_{\Psi_1}(m)\Pi_{\Psi_2}(m)$ where

$$\Psi_1 = \left\{1\left(yH'(x) \le 2\theta_0\right) : H' \in \mathcal{H}'\right\}$$

and

$$\Psi_2 = \left\{1\left(\tfrac{1}{N}|\{i : |h'_i(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\right) : H' \in \mathcal{H}'\right\}.$$

Furthermore, $\Pi_{\Psi_1}(m)$ is bounded above by the product of the growth functions for each of the N thresholded h'_is. For each i such that $\lambda_{l_i} = 1$, the growth function is bounded above by the product of the growth functions of the N_{l_i} choices of functions from $\mathcal{E}_{n_{l_i m_{ij}}}$. For each i such that $\lambda_{l_i} = 0$, the growth function is bounded above by the product of the growth function of a fixed-threshold perceptron with T_{l_i} inputs and the product of the growth functions of the T_{l_i} choices of functions from $\mathcal{E}_{n_{l_i s}}$. This gives,

$$\Pi_{\Psi_1}(m) \le \prod_{i=1}^{N}\left(\prod_{j=1}^{N_{l_i}}\Pi_{\mathcal{E}_{n_{l_i m_{ij}}}}(m)\right)^{\lambda_{l_i}}\left((em)^{T_{l_i}}\prod_{s=1}^{T_{l_i}}\Pi_{\mathcal{E}_{n_{l_i s}}}(m)\right)^{(1-\lambda_{l_i})},$$

where we have used Theorem 13.9 in [20] and Sauer's Lemma to bound the growth function of a fixed-threshold perceptron. $\Pi_{\Psi_2}(m)$ is bounded similarly, except it only depends on the product of the growth functions of the thresholded h'_is such that $\lambda_{l_i} = 1$. For i such that $\lambda_{l_i} = 0$, $|h'_i| \not< \frac{1}{2}^{j_i-1}\lambda_{l_i}$. Hence, we can bound the growth function of Ψ by

$$\Pi_\Psi(m) \le \prod_{i=1}^{N}\left(\prod_{j=1}^{N_{l_i}}\Pi_{\mathcal{E}_{n_{l_i m_{ij}}}}(m)\right)^{2\lambda_{l_i}}\left((em)^{T_{l_i}}\prod_{s=1}^{T_{l_i}}\Pi_{\mathcal{E}_{n_{l_i s}}}(m)\right)^{(1-\lambda_{l_i})}. \tag{3.10}$$

Combining Equations (3.7), (3.8), (3.9) and (3.10) with Lemma 1.3 gives

$$\begin{aligned}
&\mathbf{P}_{D^m}\Big(\exists F\in\mathcal{F}, N>0, N^*>0, 0<N_t\le N^*, \lambda_t\in\{0,1\}, \theta_0\in(0,1], \theta_t\in(0,1]:\\
&\qquad \mathbf{P}_D(yF(x)\le 0) > 2\mathbf{P}_S\Big(yF(x)\le 4\theta_0 \vee \sum_{t=1}^{T}\{w_t : |f_t(x)|<5\theta_t\lambda_t\} > \theta_0/2\Big)\\
&\qquad\qquad +\mathbf{E}_G\mathbf{E}_H c + 5e^{-\theta_0^2 N/2} + \mathbf{E}_G 10\textstyle\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)\\
&\le \sum_{N=1}^{\infty}\sum_{N^*=1}^{\infty} T^N {N^*}^N 2^N \max_{l_i,N_{l_i},\lambda_{l_i}}\prod_{i=1}^{N}(T_{l_i}r)^{\lambda_{l_i}N_{l_i}} r^{(1-\lambda_{l_i})T_{l_i}} 2(N+2)\max_{m_{ij},n_{l_is},\theta_0}\\
&\qquad \cdot 4\prod_{i=1}^{N}\left(\prod_{j=1}^{N_{l_i}}\Pi_{\mathcal{E}_{n_{l_i}m_{ij}}}(2m)\right)^{2\lambda_{l_i}}\left((2em)^{T_{l_i}}\prod_{s=1}^{T_{l_i}}\Pi_{\mathcal{E}_{n_{l_is}}}(2m)\right)^{(1-\lambda_{l_i})}\\
&\qquad \cdot\sum_{q=1}^{N}\sum_{j_q\in I_q} e^{-mc\left(\frac{1}{2}^{j_1},\ldots,\frac{1}{2}^{j_N}\right)/8}.
\end{aligned}$$

Now if we set

$$\begin{aligned}
c=\frac{8}{m}\Bigg(&\ln(N(N+1))+\ln(N^*(N^*+1))+N\ln T+N\ln N^*+N\ln 2\\
&+\sum_{i=1}^{N}\Big[\lambda_{l_i}N_{l_i}\ln(rT_{l_i})+(1-\lambda_{l_i})T_{l_i}\ln r\Big]+\ln(2(N+2))+\ln 4\\
&+\sum_{i=1}^{N}\Bigg[2\lambda_{l_i}\sum_{j=1}^{N_{l_i}}\ln\Pi_{\mathcal{E}_{n_{l_i}m_{ij}}}(2m)+(1-\lambda_{l_i})\sum_{s=1}^{T_{l_i}}\big(\ln(2em)+\ln\Pi_{\mathcal{E}_{n_{l_is}}}(2m)\big)\Bigg]\\
&+\ln(1/\delta)+\ln N-\ln\left(\sum_{i=1}^{N}(\theta_{l_i}/2)^{\lambda_{l_i}}\right)\Bigg)
\end{aligned}\tag{3.11}$$

then

$$\begin{aligned}
&e^{-mc\left(\frac{1}{2}^{j_1},\ldots,\frac{1}{2}^{j_N}\right)/8} =\\
&\qquad \frac{1}{N(N+1)}\frac{1}{N^*(N^*+1)}T^{-N}{N^*}^{-N}2^{-N}\prod_{i=1}^{N}(rT_{l_i})^{-\lambda_{l_i}N_{l_i}}r^{-(1-\lambda_{l_i})T_{l_i}}\frac{1}{2(N+2)}\\
&\qquad \cdot\frac{1}{4}\prod_{i=1}^{N}\left(\prod_{j=1}^{N_{l_i}}\Pi_{\mathcal{E}_{n_{l_i}m_{ij}}}(2m)\right)^{-2\lambda_{l_i}}\left((2em)^{T_{l_i}}\prod_{s=1}^{T_{l_i}}\Pi_{\mathcal{E}_{n_{l_is}}}(2m)\right)^{-(1-\lambda_{l_i})}\\
&\qquad \cdot\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}^{\lambda_{l_i}(j_i+1)}\delta,
\end{aligned}$$

so we obtain

$$\mathbf{P}_{D^m}\Big(\exists F\in\mathcal{F}, N>0, N^*>0, 0<N_t\le N^*, \lambda_t\in\{0,1\}, \theta_0\in(0,1], \theta_t\in(0,1]:$$

$$\mathbf{P}_D(yF(x) \le 0) > 2\mathbf{P}_S\Big(yF(x) \le 4\theta_0 \vee \sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} > \theta_0/2\Big)$$
$$+\mathbf{E}_G\mathbf{E}_H c + 5e^{-\theta_0^2 N/2} + \mathbf{E}_G 10\textstyle\sum_{i=1}^{N}\lambda_{l_i}e^{-{\theta_{l_i}}^2 N_{l_i}/2b^2}\Big)$$
$$\le \sum_{N=1}^{\infty}\frac{1}{N(N+1)}\sum_{N^*=1}^{\infty}\frac{1}{N^*(N^*+1)}\sum_{q=1}^{N}\sum_{j_q\in I_q}\tfrac{1}{N}\sum_{i=1}^{N}\tfrac{1}{2}^{\lambda_{l_i}(j_i+1)}\delta$$
$$= \delta.$$

Now setting

$$N = \left\lceil \frac{1}{\theta_0^2}\ln\frac{5m}{8}\right\rceil, \quad N_t = \left\lceil \frac{b^2}{\theta_t^2}\ln\frac{10Nm}{8}\right\rceil^{\lambda_t} \quad \text{and} \quad N^* = \max_t N_t,$$

gives that with probability at least $1-\delta$ over choice of the sample set S every $F \in \mathcal{F}$, $\lambda_t \in \{0,1\}$, $\theta_0 \in (0,1]$ and $\theta_t \in (0,1]$ satisfies

$$\mathbf{P}_D(yF(x) \le 0) \le 2\mathbf{P}_S\Big(yF(x) \le 4\theta_0 \vee \sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} > \theta_0/2\Big)$$
$$+\mathbf{E}_G\mathbf{E}_H c + \frac{16}{m}. \tag{3.12}$$

Notice that

$$-\ln\left(\sum_{i=1}^{N}(\theta_{l_i}/2)^{\lambda_{l_i}}\right) \ge -\sum_{i=1}^{N}\ln\left((\theta_{l_i}/2)^{\lambda_{l_i}}\right) = \sum_{i=1}^{N}\lambda_{l_i}\ln\left(2/\theta_{l_i}\right),$$

where the first inequality follows from the fact that the sum of a sequence of $(0,1]$-valued variables is always larger than the product of those variables. By observing that

$$\mathbf{E}_G\mathbf{E}_H\sum_{i=1}^{N}\lambda_{l_i}\ln\left(2/\theta_{l_i}\right) = N\sum_{t=1}^{T}w_t\lambda_t\ln\left(2/\theta_t\right),$$

$$\mathbf{E}_G\mathbf{E}_H\sum_{i=1}^{N}\Big[\lambda_{l_i}N_{l_i}\ln(rT_{l_i}) + (1-\lambda_{l_i})T_{l_i}\ln r\Big] = N\sum_{t=1}^{T}w_t\Big[\lambda_t N_t\ln(rT_t) + (1-\lambda_t)T_t\ln r\Big]$$

and

$$\mathbf{E}_G\mathbf{E}_H\sum_{i=1}^{N}\left[2\lambda_{l_i}\sum_{j=1}^{N_{l_i}}\ln\Pi_{\mathcal{E}_{n_{l_i m_{ij}}}}(2m) + (1-\lambda_{l_i})\sum_{s=1}^{T_{l_i}}\big(\ln(2em) + \ln\Pi_{\mathcal{E}_{n_{l_i s}}}(2m)\big)\right]$$
$$= N\sum_{t=1}^{T}w_t\left[2\lambda_t N_t\sum_{s=1}^{T_t}v_{ts}\ln\Pi_{\mathcal{E}_{n_{ts}}}(2m) + (1-\lambda_t)\sum_{s=1}^{T_t}\big(\ln(2em) + \ln\Pi_{\mathcal{E}_{n_{ts}}}(2m)\big)\right],$$

we obtain

$$\begin{aligned}\mathbf{E}_G\mathbf{E}_{HC} \leq \frac{8}{m}\Bigg(& \ln(N(N+1)) + \ln(N^*(N^*+1)) + N\ln T + N\ln N^* + N\ln 2 \\ & + N\sum_{t=1}^{T} w_t\Big[\lambda_t N_t \ln(rT_t) + (1-\lambda_t)T_t \ln r\Big] + \ln(2(N+2)) + \ln 4 \\ & + N\sum_{t=1}^{T} w_t\Bigg[2\lambda_t N_t \sum_{s=1}^{T_t} v_{ts} \ln \Pi_{\mathcal{E}_{n_{ts}}}(2m) \\ & \qquad + (1-\lambda_t)\sum_{s=1}^{T_t}\big(\ln(2em) + \ln \Pi_{\mathcal{E}_{n_{ts}}}(2m)\big)\Bigg] \\ & + \ln(1/\delta) + \ln N + N\sum_{t=1}^{T} w_t\lambda_t \ln\left(2/\theta_t\right)\Bigg).\end{aligned}$$

Substituting for N, N_t and N^*, collecting terms and simplifying gives

$$\begin{aligned}\mathbf{E}_G\mathbf{E}_{HC} \leq \kappa\frac{1}{m}\Bigg[& \frac{\ln m}{\theta_0^2}\left(\ln\left(\frac{Tb^2\ln(m/\theta_0)}{\min_t \theta_t^2}\right)\right. \\ & + \sum_{t=1}^{T} w_t\Bigg[\lambda_t \frac{b^2\ln(m/\theta_0)}{\theta_t^2}\left(\ln(rT_t) + \sum_{s=1}^{T_t} v_{ts}\ln \Pi_{\mathcal{E}_{n_{ts}}}(2m)\right) \\ & \left.\left. + (1-\lambda_t)\left(T_t\ln(rm) + \sum_{s=1}^{T_t}\ln \Pi_{\mathcal{E}_{n_{ts}}}(2m)\right)\Bigg]\right) + \ln(1/\delta)\Bigg], \qquad (3.13)\end{aligned}$$

for some constant κ. Some of the simplifications applied to obtain (3.13) means that the result only holds for $b \geq 1$, $T > 1$ and $m > 1$. All of these are trivial restrictions.

Decomposing the first term on the right hand side of (3.12) via the union bound gives

$$\begin{aligned}2\mathbf{P}_S\Big(yF(x) \leq 4\theta_0 \vee \sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} > \theta_0/2\Big) & \\ \leq 2\mathbf{P}_S\Big(yF(x) \leq 4\theta_0\Big) + 2\mathbf{P}_S\Big(\sum_{t=1}^{T}\{w_t : |f_t(x)| < 5\theta_t\lambda_t\} > \theta_0/2\Big) & \\ \leq 2\mathbf{P}_S\big(yF(x) \leq 4\theta_0\big) + \frac{4}{\theta_0}\sum_{t=1}^{T} w_t\mathbf{P}_S\big(|f_t(x)| < 5\theta_t\lambda_t\big), & \qquad (3.14)\end{aligned}$$

where we have used Markov's inequality ($\mathbf{P}(Z > \alpha) \leq \mathbf{E}(Z)/\alpha$ for a nonnegative random variable Z) to bound the second term.

Hence, there exists a constant κ such that with probability at least $1-\delta$ over random choice of the sample set S every $F \in \mathcal{F}$, $\lambda_t \in \{0,1\}$, $\theta_0 \in (0,1]$ and $\theta_t \in (0,1]$ satisfies

$$\mathbf{P}_D(yF(x) \leq 0) \leq 2\mathbf{P}_S(yF(x) \leq 4\theta_0)$$

$$+\kappa\frac{1}{m}\left[\frac{\ln m}{\theta_0^2}\left(\sum_{t=1}^{T} w_t(\lambda_t C_1 + (1-\lambda_t)C_2) + C_3\right) + \ln(1/\delta)\right],$$

where

$$\begin{aligned} C_1 &= \frac{b^2\ln(m/\theta_0)}{\theta_t^2}\left(\ln(rT_t) + \sum_{s=1}^{T_t} v_{ts}\ln\Pi_{\mathcal{E}_{n_{ts}}}(2m)\right) + \frac{m\theta_0}{\ln m}\mathbf{P}_S\big(|f_t(x)| < 5\theta_t\big), \\ C_2 &= T_t\ln(rm) + \sum_{s=1}^{T_t}\ln\Pi_{\mathcal{E}_{n_{ts}}}(2m), \\ C_3 &= \ln\left(\frac{Tb^2\ln(m/\theta_0)}{\min_t\theta_t^2}\right). \end{aligned}$$

Finally, in order to obtain the smallest bound over choice of the λ_t we set $\lambda_t = 0$ when $C_1 > C_2$ and $\lambda_t = 1$ otherwise. Substituting $\theta_0/4$ for θ_0 and $\theta_t/5$ for θ_t then gives the required result. □

There are two things to note about this proof. Firstly, in order to simplify the details of the argument, little effort was made in optimizing the choice of constants. In particular, the choice of constants in R could be improved so that the both uses of θ_0 are arbitrarily close to being within a factor of 2, at the expense of worse constants in the exponential terms in Lemma 3.3. Secondly, to improve the readability of the final result, some of the log terms are larger than they need to be.

3.3.2 Proof of Theorem 3.2

Proof

Recall that

$$f_t(x) = \sum_{s=1}^{T_t} v_{ts}\sigma_{ts}L_{ts}(x),$$

and that $e_{ts}(x) = \sigma_{ts}L_{ts}(x)$. Setting $r = \max_t d_t$ (recalling that d_t is the depth of tree t), for each $n = 1, \ldots, r$ we define the class $\mathcal{E}_n$ by

$$\mathcal{E}_n = \{\sigma L : \sigma \in \{\pm 1\}, L \in \mathcal{B}_n\},$$

where $\mathcal{B}_n$ is the class of leaf functions for leaves up to depth n defined by

$$\mathcal{B}_n = \{L : L = u_1 \wedge u_2 \wedge \cdots \wedge u_v | v \leq w, u_1, \ldots, u_v \in \mathcal{U}\},$$

where $\mathcal{U}$ denotes the class of node decision functions as before. So we are separating the leaf functions into d classes according to their depth. Hence,

$$\begin{aligned}\Pi_{\mathcal{E}_n}(m) &= \max\left\{\left|\left\{\left(\sigma L(x_1),\ldots,\sigma L(x_m)\right) : \sigma\in\{\pm 1\}, L\in\mathcal{B}_n\right\}\right| : x_1,\ldots,x_m\in X\right\}\\ &\leq 2\prod_{l=1}^{n}\max\left\{\left|\left\{\left(u(x_1),\ldots,u(x_m)\right) : u\in\mathcal{U}\right\}\right| : x_1,\ldots,x_m\in X\right\}\\ &= 2\left(\Pi_{\mathcal{U}}(m)\right)^n\end{aligned}$$

and thus

$$\sum_{s=1}^{T_t} v_{ts}\ln\left(\Pi_{\mathcal{E}_{n_{ts}}}(2m)\right) \leq 2\sum_{s=1}^{T_t} v_{ts}d_{ts}\ln\Pi_{\mathcal{U}}(2m) = 2\bar{d}_t\ln\Pi_{\mathcal{U}}(2m), \tag{3.15}$$

since d_{ts} denotes the depth of leaf s in tree t.

The proof follows that of Theorem 3.1 until (3.10) where we give an upper bound on $\Pi_\Psi(m)$ for

$$\Psi = \left\{1\left(yH'(x)\leq 2\theta_0 \vee \tfrac{1}{N}|\{i : |h_i'(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\right) : H'\in\mathcal{H}'\right\}.$$

For the special case when h_i' is a decision tree we can do considerably better than (3.10). As before, we note that $\Pi_\Psi(m)$ is bounded above by $\Pi_{\Psi_1}(m)\Pi_{\Psi_2}(m)$ where

$$\Psi_1 = \left\{1\left(yH'(x)\leq 2\theta_0\right) : H'\in\mathcal{H}'\right\}$$

and

$$\Psi_2 = \left\{1\left(\tfrac{1}{N}|\{i : |h_i'(x)| < \tfrac{1}{2}^{j_i-1}\lambda_{l_i}\}| > \theta_0\right) : H'\in\mathcal{H}'\right\}.$$

Again, $\Pi_{\Psi_1}(m)$ is bounded above by the product of the growth functions for each of the N thresholded h_i's. For each i such that $\lambda_{l_i}=1$, the growth function is bounded above by the product of the growth functions of the N_{l_i} choices of functions from $\mathcal{E}_{n_{l_i m_{ij}}}$. However, for each i such that $\lambda_{l_i}=0$, we notice that the growth function of the thresholded h_i' is simply the growth function of a decision tree with T_{l_i} leaves. This gives,

$$\Pi_{\Psi_1}(m)\leq\prod_{i=1}^{N}\left(\prod_{j=1}^{N_{l_i}}\Pi_{\mathcal{E}_{n_{l_i m_{ij}}}}(m)\right)^{\lambda_{l_i}}\left(\left(\Pi_{\mathcal{U}}(m)+2\right)^{2T_{l_i}-1}\right)^{(1-\lambda_{l_i})},$$

where $\mathcal{U}$ denotes the class of node decision functions. We have bounded the growth function of a decision tree by the product of the growth functions of the $2T_{l_i}-1$ nodes (made up of T_{l_i} $\{\pm 1\}$-valued leaves and $T_{l_i}-1$ internal nodes whose decisions come from some class $\mathcal{U}$). This gives an upper bound on the growth function since we can represent

a decision tree by listing, in breadth-first order, the node functions and leaf labels. Now, applying the same upper bound for $\Pi_{\Psi_2}(m)$ as before gives that the growth function of Ψ is bounded by

$$\Pi_\Psi(m) \leq \prod_{i=1}^{N} \left(\prod_{j=1}^{N_{l_i}} \Pi_{\mathcal{E}_{n_{l_i m_{ij}}}}(m) \right)^{2\lambda_{l_i}} \left(\left(\Pi_{\mathcal{U}}(m)+2\right)^{2T_{l_i}-1} \right)^{(1-\lambda_{l_i})}.$$

We now proceed exactly as in the proof of Theorem 3.1 except

$$(1-\lambda_{l_i}) \sum_{s=1}^{T_{l_i}} \left(\ln(2em) + \ln \Pi_{\mathcal{E}_{n_{l_i s}}}(2m) \right)$$

is replaced with

$$(1-\lambda_{l_i})(2T_{l_i}-1) \ln\left(\Pi_{\mathcal{U}}(2m)+2\right)$$

in (3.11) where we choose c. From this we obtain the following result. There exists a constant κ such that with probability at least $1-\delta$ over random choice of the sample set S, every convex combination F of decision trees, $\theta_0 \in (0,1]$ and $\theta_t \in (0,1]$ satisfies

$$\begin{aligned} \mathbf{P}_D(yF(x) \leq 0) \quad &\leq \quad 2\mathbf{P}_S(yF(x) \leq \theta_0) \\ &+ \quad \kappa \frac{1}{m} \left[\frac{\ln m}{\theta_0^2} \left(\sum_{t=1}^{T} w_t \min(C_1, C_2) + C_3 \right) + \ln(1/\delta) \right], \end{aligned} \tag{3.16}$$

where

$$\begin{aligned} C_1 &= \frac{\ln(m/\theta_0)}{\theta_t^2} \left(\ln(T_t \max_l d_l) + \sum_{s=1}^{T_t} v_{ts} \ln\left(\Pi_{\mathcal{E}_{n_{ts}}}(2m)\right) \right) + \frac{m\theta_0}{\ln m} \mathbf{P}_S\left(|f_t(x)| < \theta_t\right), \\ C_2 &= T_i \Big(\ln(\max_l d_l) + \ln\left(\Pi_{\mathcal{U}}(2m)+2\right) \Big), \\ C_3 &= \ln\left(\frac{T \ln(m/\theta_0)}{\min_t \theta_t^2} \right). \end{aligned}$$

To obtain this result we have set $b=1$ (which follows from the definition of the $\mathcal{E}_n$) and substituted for $r = \max_l d_l$. Lastly, we note that by definition of f_t,

$$\begin{aligned} \mathbf{P}_S(|f_t(x)| < \theta_t) &= \mathbf{P}_S\left(\left| \sum_{s=1}^{T_t} v_{ts}\sigma_{ts}L_{ts}(x) \right| < \theta_t \right) \\ &= \mathbf{P}_S\left(v_{ts} < \theta_t \text{ for the } s \text{ such that } L_{ts}(x)=1\right) \\ &= \sum_{s=1}^{T_t} P_{ts} 1(v_{ts} < \theta_t), \end{aligned} \tag{3.17}$$

where, for a predicate A, $1(A)=1$ if A is true and 0 otherwise. The second equality follows from the fact that for each value of x, $L_{ts}(x)=0$ for all values of s except the

one corresponding to the leaf which is used to classify x.

To complete the proof we substitute (3.15) and (3.17) into (3.16), apply Sauer's Lemma to bound $\Pi_{\mathcal{U}}(m)$ in terms of $\text{VCdim}(\mathcal{U})$ and simplify. □

3.4 Discussion

Although we only applied our general result of Section 3.1 to give upper bounds on the generalization error of voted combinations of decision trees, there are other classes of classifiers to which this result can easily be applied. We mention two such classes here.

By considering a single hidden-layer threshold network as a convex combination of single perceptrons we can obtain a bound on the generalization error of such networks in terms of the margin and the average complexity of the perceptrons (where the average is in terms of the perceptron weights in the network). The complexity of each perceptron in this result is related to the proportion of training examples which are close to the perceptron threshold. The measure of complexity suggested by existing VC bounds for threshold networks (see, for example, [10]) is related to the number of weights in the network. If a network classifies most examples with a large margin and the network's perceptrons have few examples close to threshold, then our measure of complexity can be considerably smaller. For details see [50].

A binary mask perceptron, as introduced by Kowalczyk and Ferrá in [45], is a thresholded combination of functions which consist of products of $\{0,1\}$-valued decision functions from some class $\mathcal{U}$. We can thus apply our general result to obtain a bound on the generalization error of a voted combination of binary mask perceptrons in terms of the margin and the average complexity of the mask perceptrons (where the average is in terms of the voting weights assigned to the mask perceptrons). The complexity of each mask perceptron in this result is related to the to the proportion of training examples which are close to the mask perceptron threshold. For details see [49].

CHAPTER

4 The alternating decision tree

Part of the popularity of algorithms for constructing decision trees (such as CART [16] and C4.5 [54]) is due to the fact that they produce classifiers which are highly accurate, yet still admit simple interpretation. The application of voting methods, particularly the AdaBoost algorithm, to the construction of voted combinations of decision trees has proven to be remarkably successful [13, 34, 37, 55]. Although these classifiers are generally significantly more accurate than a single decision tree, they are typically far more complex and difficult to interpret.

In Chapters 2 and 3 we showed that existing techniques for the analysis of thresholded real-valued classifiers can be applied further afield. By representing a decision tree as a thresholded convex combination of leaf functions we derived upper bounds on the generalization error of decision trees and voted combinations of decision trees.

In Section 4.1 of this chapter we show that by representing a decision tree as a more general sort of voted combination we can obtain a new type of classifier, the *alternating decision tree*, which provides a natural generalization of decision trees and voted combinations of decision trees. We then show that alternating decision trees, by virtue of their representation as a voted combination, can be constructed directly using existing voting methods. In particular, in Section 4.2 we present an algorithm for learning alternating decision trees based upon Schapire and Singer's real-valued extension of AdaBoost [61].

Despite their generality, alternating decision trees can be interpreted using the same simple techniques that are applied to interpret single decision trees and voted combinations of decision stumps. In Section 4.3 we demonstrate these techniques by analyzing the alternating decision tree generated using our algorithm for a data set from the UC Irvine repository.

Finally, in Section 4.4 we present some experiments using our algorithm on a selection of data sets taken from the UC Irvine repository. These

experiments show that our algorithm for building alternating decision trees is competitive with algorithms for building voted combinations of decision trees, such as C5.0.

4.1 Generalizing decision trees

We begin this section by describing a new representation for decision trees and then show how this representation can be extended to describe alternating decision trees. In Chapters 2 and 3 we showed that a decision tree can be represented as a voted combination of its leaf functions. In a similar manner, we will show how alternating decision trees can also be represented as a voted combination of simple functions.

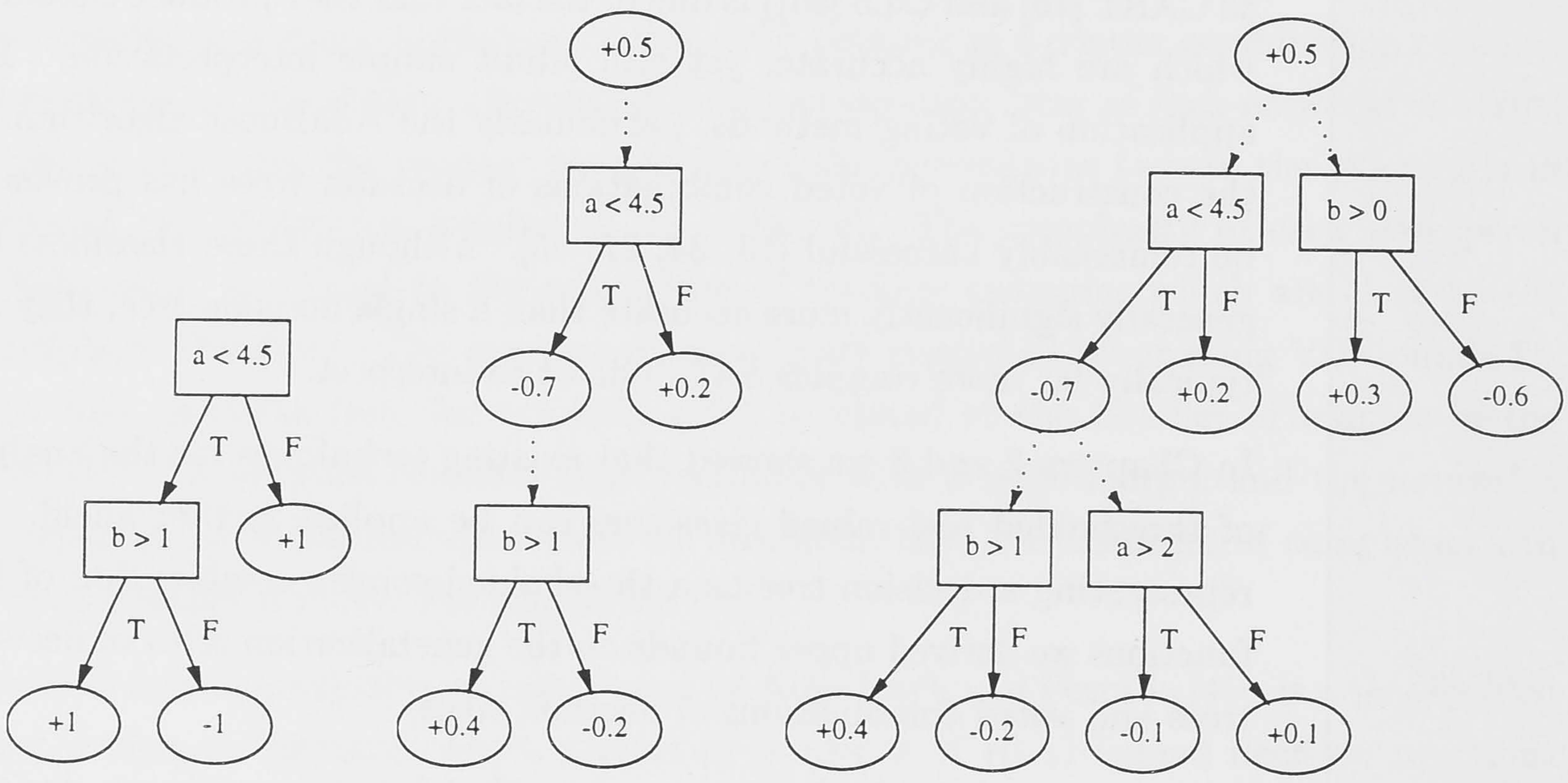


Figure 4.1: Tree-based classifiers: a decision tree (left), the same decision tree represented as an alternating decision tree (middle), and a general alternating decision tree (right).

Let us consider the simple decision tree shown in the left of Figure 4.1. This tree has two decision nodes and three leaves. We call these leaf nodes *prediction nodes* because they are used to give the output prediction of the tree, a notion we will generalize for alternating decision trees. The tree maps a measurement $(a, b) \in \mathbb{R}^2$ to either $+1$ or -1. An alternative representation of this decision tree is shown in the middle of Figure 4.1. This tree also has two decision nodes, but has five prediction nodes, which are not all leaves and are now associated with a real-valued number rather than $+1$ or -1. In these sort of diagrams we will use squares to denote *decision nodes* and ellipses to denote *prediction nodes*. For the tree shown in the middle of Figure 4.1, a measurement is classified like a decision tree, by following the *path* in the tree from the root to one of the leaves corresponding to the evaluation of decision nodes along the way. However, *unlike* a decision tree, the classification is *not* the label of the leaf. Instead, the classification is the sign of the sum of the values of the prediction nodes

along the path from the root to the leaf. For example, the classification of $(a, b) = (0, 0)$ is $\text{sign}(0.5 - 0.7 - 0.2) = \text{sign}(-0.4) = -1$. It can easily be verified that this tree gives the same output as the previous decision tree. It is also clear that many different trees of this new type can represent the same decision tree.

We call this new type of tree an *alternating decision tree* because it consists of alternating layers of prediction nodes and decision nodes. As stated in Chapter 1, a decision tree defines a partition of the measurement space into disjoint regions. Most decision tree algorithms recursively subdivide the measurement space by splitting existing nodes. In a decision tree nodes can be split at most once. No such requirement exists for an alternating decision tree. This multiple splitting corresponds, in an alternating decision tree, to a prediction node with more than one child decision node. Consider our example alternating decision tree shown in the middle of Figure 4.1. Each prediction node in this tree has at most one child decision node. By adding two extra decision nodes to this tree we obtain the general alternating decision tree shown in the right of Figure 4.1.

In this general alternating decision tree a measurement is classified by following a *multi-path* in the tree from the root to the leaves. As in a decision tree, when a measurement reaches a decision node it passes to the child node corresponding to the outcome of the decision associated with that node. However, when a measurement reaches a prediction node it passes to *all* of the child decision nodes. Thus, the path becomes the union of a set of paths or a *multi-path*. As before, the final classification is the sign of the sum of the values of all of the prediction nodes visited. For example, for the tree shown in the right of Figure 4.1, the classification of $(a, b) = (0, 0)$ is $\text{sign}(0.5 - 0.7 - 0.2 + 0.1 - 0.6) = \text{sign}(-0.9) = -1$.

We now give a more formal definition of an alternating decision tree as a voted combination of decision node functions. We define a *base condition* to be a Boolean predicate over elements of X. We use $\wedge$ to denote conjunction, $\neg$ to denote negation and $\mathbf{T}$ to denote the constant predicate which always evaluates true. We use $\mathcal{C}$ to denote a set of base conditions. For our purposes we only consider $\mathcal{C}$ to consist of tests of the form 'is $x^k > a$' for $x \in \mathbb{R}^n$, $k \in \{1, \ldots, n\}$ and $a \in \mathbb{R}$. Tests associated with decision nodes correspond to base conditions. We define a *precondition* to be a conjunction of base conditions and negations of base conditions. Sequences of tests along a path in the tree correspond to preconditions.

An alternating decision tree classifier is a function $\text{sign}(f_0 + \sum_{t=1}^{T} f_t(x))$, where f_0 is a real constant and f_t is defined by

$$f_t(x) = \begin{cases} \alpha_t & : \text{ if } x \text{ satisfies } p_t \wedge c_t \\ \beta_t & : \text{ if } x \text{ satisfies } p_t \wedge \neg c_t \\ 0 & : \text{ if } x \text{ satisfies } \neg p_t \end{cases}$$

where α_t and β_t are real constants, p_t is a precondition and c_t is a base condition in $\mathcal{C}$. The constant f_0 corresponds to the value associated with the root prediction node. Each f_t corresponds to a decision node in the tree where p_t represents the sequence of tests along the path from the root to the node and c_t represents the test associated with the node. The constants α_t and β_t are the values associated with the true and false child prediction nodes respectively.

For the alternating decision tree defined in this way to be valid each decision node f_t must satisfy the following property: for every t, either $p_t = \mathbf{T}$, or for some $s \neq t$, $p_t = p_s \wedge c_s$ or $p_t = p_s \wedge \neg c_s$. If $p_t = \mathbf{T}$, then f_t corresponds to a decision node which is a child of the root prediction node. If $p_t = p_s \wedge c_s$, then f_t corresponds to a decision node which is a child of the true child prediction node of the decision node corresponding to f_s. Similarly, if $p_t = p_s \wedge \neg c_s$, then f_t corresponds to a decision node which is a child of the false child prediction node of the decision node corresponding to f_s.

As an example, let us consider the alternating decision tree shown in the middle of Figure 4.1. For this tree, $f_0 = 0.5$ and the two decision nodes are represented as

$$f_1(x) = \begin{cases} -0.7 & : \text{ if } \mathbf{T} \wedge (a < 4.5) \\ +0.2 & : \text{ if } \mathbf{T} \wedge \neg(a < 4.5) \\ 0 & : \text{ if } \neg\mathbf{T} \end{cases}$$

and

$$f_2(x) = \begin{cases} +0.4 & : \text{ if } (a < 4.5) \wedge (b > 1) \\ -0.2 & : \text{ if } (a < 4.5) \wedge \neg(b > 1) \\ 0 & : \text{ if } \neg(a < 4.5) \end{cases}.$$

From these definitions it can be seen that $p_1 = \mathbf{T}$, $c_1 = (a < 4.5)$, $p_2 = (a < 4.5)$ and $c_2 = (b > 1)$. The output from this tree is then $\text{sign}(f_0 + f_1(x) + f_2(x))$.

From our description it is clear that the alternating decision tree is a generalization of the decision tree. The alternating decision tree also generalizes voted combinations of decision stumps and voted combinations of decision trees. Voted combinations of decision stumps can be represented by an alternating decision tree with exactly three layers: a root prediction node, a set of decision nodes, and the prediction nodes associated with these decision nodes. Voted combinations of decision trees can be obtained by restricting the alternating decision tree so that only the root prediction node has more than one child decision node.

The alternating decision tree described in this section is similar to the *option tree* first described by Buntine in [17] and further developed by Kohavi *et al.* in [44]. Option trees were shown to provide significant improvements in generalization error compared

to single decision trees. The results reported in [44] are comparable to voted combinations of decision trees generated by Bagging [13]. As we shall see in Section 4.4, the algorithm we describe in the next section for building alternating decision trees achieves better performance levels, comparable to voted combinations of decision trees generated by AdaBoost.

4.2 Algorithm

In the previous section we gave a formal definition of the alternating decision tree as a voted combination of decision node functions. Given this representation, it is a simple matter to modify an existing voting method so that it can be used to construct alternating decision trees. The only unusual feature of the resulting algorithm is that the class of base classifiers consists of the possible decision nodes (and their child predictions) which can be added to the tree. This means that the class of base classifiers grows as the tree is constructed.

Since our predictions are real-valued, instead of using the original AdaBoost algorithm described in Chapter 1, we use Schapire and Singer's real-valued extension of AdaBoost [61]. Like the original, the real-valued version maintains a probability distribution D_t over training examples at each round. However, instead of selecting base classifiers at each round to minimize the weighted training error relative to D_t, the algorithm chooses base classifiers at each round to minimize

$$\sum_{i=1}^{m} D_t(i) e^{-y_i f_t(x_i)}. \tag{4.1}$$

As shown in the previous section, each decision node f_t in the tree can be viewed as a real-valued classifier which partitions the measurement space into three regions corresponding to $p_t \wedge c_t$ (examples which evaluate true at the decision node), $p_t \wedge \neg c_t$ (examples which evaluate false at the decision node) and $\neg p_t$ (examples which do not reach the decision node). The first two of these regions are associated with the real-valued classification corresponding to the true and false child prediction node values respectively, while the last region is associated with a prediction of zero.

For base classifiers which partition the measurement space, Schapire and Singer give formulas for calculating the best predictions for a given partition in order to minimize (4.1). In our case, these formulas reduce to choosing f_t at each round to minimize

$$2\left(\sqrt{W_t^+(p_t \wedge c_t)W_t^-(p_t \wedge c_t)} + \sqrt{W_t^+(p_t \wedge \neg c_t)W_t^-(p_t \wedge \neg c_t)}\right) + W_t(\neg p), \tag{4.2}$$

and choosing predictions for the regions corresponding to $p_t \wedge c_t$ and $p_t \wedge \neg c_t$ to be

$$\tfrac{1}{2} \ln \left(\frac{W_t^+(p_t \wedge c_t)}{W_t^-(p_t \wedge c_t)} \right)$$

and

$$\tfrac{1}{2} \ln \left(\frac{W_t^+(p_t \wedge \neg c_t)}{W_t^-(p_t \wedge \neg c_t)} \right)$$

respectively. Here we use $W_t(c)$ to denote the total weight (relative to D_t) of training examples which satisfy a predicate c. Similarly, $W_t^+(c)$ and $W_t^-(c)$ denote the total weight of training examples which satisfy c and are labelled $+1$ and -1 respectively. Hence, $W_t(c) = W_t^+(c) + W_t^-(c)$.

Algorithm 4.1: ADTree

Require :

- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in \mathbb{R}^n \times \{\pm 1\}$.
- A set of base conditions $\mathcal{C}$ (tests of the form 'is $x^k > a$' for $x \in \mathbb{R}^n$, $k \in \{1, \ldots, n\}$ and $a \in \mathbb{R}$).

let $D_0(i) := 1/m$ for $i = 1, \ldots, m$

let $f_0 := \dfrac{1}{2} \ln \left(\dfrac{W_0^+(\mathbf{T})}{W_0^-(\mathbf{T})} \right)$

let $F_0 := f_0$

let $\mathcal{P}_1 := \{\mathbf{T}\}$

for $t := 1$ **to** T

let $D_t(i) := D_{t-1}(i) \exp(-y_i f_{t-1}(x_i))/Z_t$ for $i = 1, \ldots, m$ (where Z_t normalizes D_t)

$$\textbf{let } (p_t, c_t) = \underset{(p \in \mathcal{P}_t, c \in \mathcal{C})}{\operatorname{argmin}} \; 2\left(\sqrt{W_t^+(p \wedge c) W_t^-(p \wedge c)} + \sqrt{W_t^+(p \wedge \neg c) W_t^-(p \wedge \neg c)} \right) + W_t(\neg p)$$

$$\textbf{let } f_t(x) := \begin{cases} \frac{1}{2} \ln \left(\dfrac{W_t^+(p_t \wedge c_t)}{W_t^-(p_t \wedge c_t)} \right) & : \text{ if } x \text{ satisfies } p_t \wedge c_t \\ \frac{1}{2} \ln \left(\dfrac{W_t^+(p_t \wedge \neg c_t)}{W_t^-(p_t \wedge \neg c_t)} \right) & : \text{ if } x \text{ satisfies } p_t \wedge \neg c_t \\ 0 & : \text{ if } x \text{ satisfies } \neg p_t \end{cases}$$

let $F_t := F_{t-1} + f_t$

let $\mathcal{P}_{t+1} := \mathcal{P}_t \cup \{p_t \wedge c_t, p_t \wedge \neg c_t\}$

return F_T

The resulting algorithm for building alternating decision trees, *ADTree*, is shown in Algorithm 4.1. The algorithm starts by finding the best constant prediction for the entire training set. This prediction is placed at the root of the tree. The algorithm then iteratively adds decision nodes (and their child predictions) to the tree.

The algorithm maintains a set $\mathcal{P}_t$ of preconditions at each round. The preconditions in this set represent the paths of all of the prediction nodes in the tree at round t. At each round, the algorithm evaluates (4.2) for every possible split (as represented by elements of $\mathcal{C}$) of every prediction node currently in the tree (as represented by elements of $\mathcal{P}_t$) and chooses the split with smallest value. The decision node corresponding to this split of this prediction node is then added to the tree. The distribution over training examples is then updated to emphasize examples which the newly added decision node classifies incorrectly.

In [42], Kearns and Mansour analyze decision tree learning algorithms in terms of boosting. Their analysis suggests an algorithm similar to the one presented here.

4.3 Interpreting alternating decision trees

In this section we demonstrate how alternating decision trees can be interpreted using the same techniques that are applied to interpret decision trees and voted decision stumps. The question of improving the comprehensibility of trained classifiers has been previously studied by several authors [19, 24, 36, 48]. In particular, in [36], Friedman describes several techniques for interpreting voted combinations of decision trees.

Throughout this section we will refer to the alternating decision tree shown in Figure 4.2 which was generated for the `cleve` data set. This data set consists of medical diagnostics for a selection of patients tested for heart disease. Positive output corresponds to 'healthy' and negative to 'sick'. This tree was generated using the ADTree algorithm described in the previous section. The alternating decision tree consists of six decision nodes and has an average test error of 17%. As we shall see later, the voted combination of decision trees generated by C5.0 for this data set has an average test error of 20% and consists of over 400 decision nodes!

The fact that the alternating decision tree is so much smaller already makes for easier interpretation. However, the main reason that alternating decision trees are easy to interpret is that the contribution of each decision node in the tree can be understood in isolation. The sum of these contributions gives the prediction and hence the classification. For example, for the tree in Figure 4.2, the `sex = female` decision node indicates that being female is a strong indicator against heart disease (since it gives a large positive contribution to the prediction). On the other hand, the `number-vessels-colored = 0` decision node indicates that having colored vessels is a strong indicator for heart disease

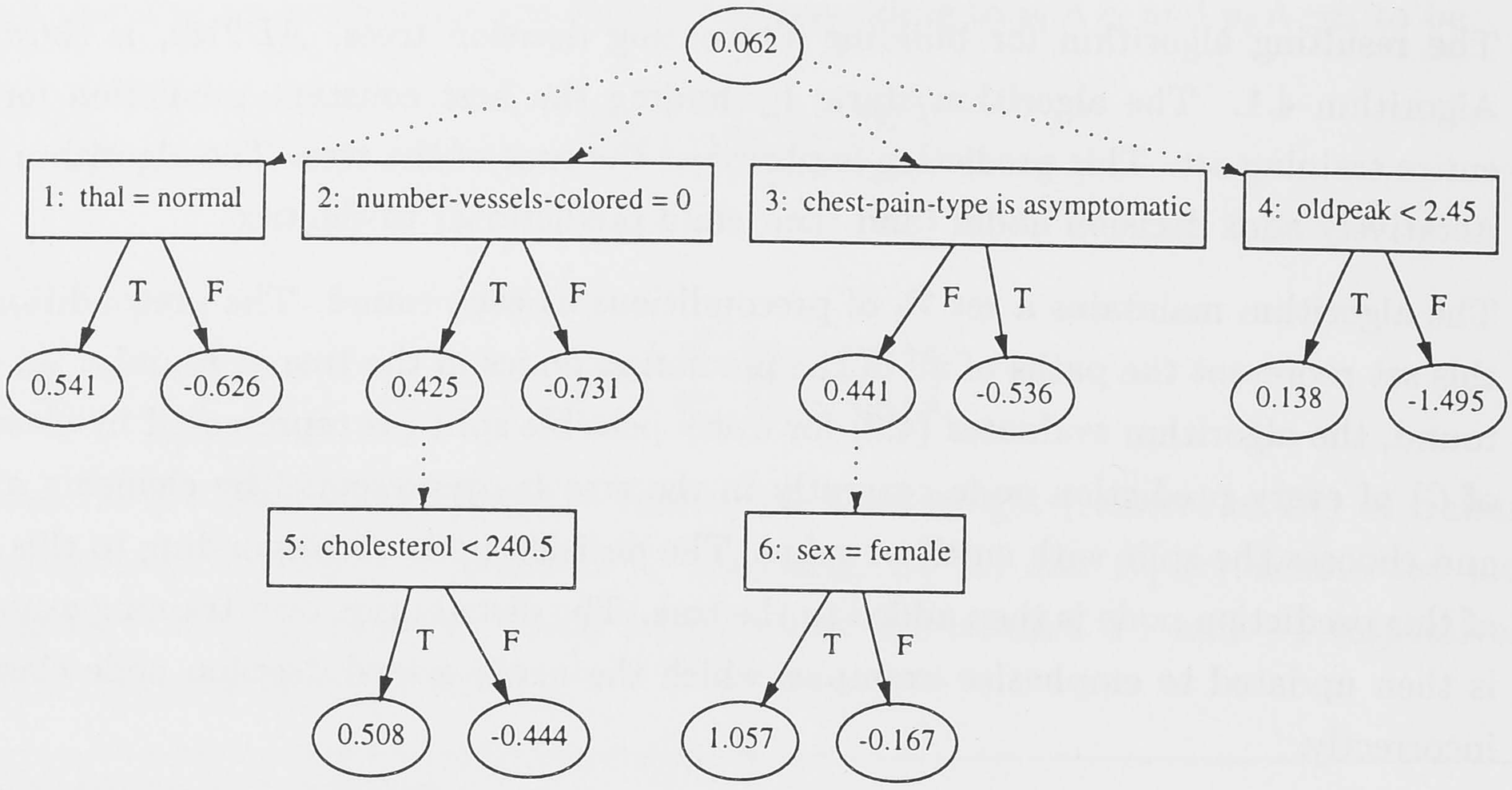


Figure 4.2: An alternating decision tree for the `cleve` data set (negative numbers correspond to 'sick' and positive to 'healthy').

(since it gives a large negative contribution to the prediction). All of the decision nodes in the tree can be similarly interpreted.

After understanding the contribution of the decision nodes in isolation we can also analyze the interactions between the decision nodes. Parallel decision nodes, such as the four nodes in the first level, are essentially independent. For example, if the `thal = normal` decision node evaluates to true then it gives a positive contribution to the final prediction irrespective of whether the `chest-pain-type is asymptomatic` decision node evaluates true or false. This relationship is the same as that represented by nodes in a voted combination of decision stumps. This sort of relationship could be represented by a standard decision tree, but the tree would need many more decision nodes.

In contrast to the parallel decision nodes in the first level, the decision nodes in the second level depend on the evaluation of their ancestral nodes. For example, the `cholesterol < 240.5` decision node depends on the `number-vessels-colored = 0` decision node evaluating true. This indicates that it is only worthwhile to check a patient's cholesterol level when they have colored vessels. The same reasoning holds for the `sex = female` decision node in relation to the `chest-pain-type is asymptomatic` decision node. This relationship is the same as that represented by nodes in a decision tree. This sort of relationship cannot be represented by a voted combination of decision stumps.

In our example the root node always contributes 0.062. This small positive number indicates that, according to this training set, there were slightly more 'healthy' cases examined than 'sick' cases. This implies that before testing anything we could predict 'healthy' with low confidence. We associate a low level of confidence with this prediction

because the absolute value of the prediction is small. If we evaluated further and any node in the tree were to contribute even a small negative value the classification would change.

The final prediction is the sum of the contributions of all of the decision nodes, and this prediction is thresholded to give the classification. By testing the conditions for each decision node in the tree in a serial fashion we can view this process as the accumulation of evidence for or against heart disease as we proceed. If at some point during this process we have a sum whose absolute value is large, and the total contribution of all of the untested nodes is small, then we don't need to continue — the current sign of the sum cannot change. Although this situation is not likely to occur in our small example tree, it is a common occurrence in larger trees. Even if it is possible for the sign to change, it may require that most of the future contributions are the opposite of the sign of the current sum, which is unlikely.

The fact that each decision node has a limited influence can be exploited in practice. In many situations we cannot evaluate all of the decision nodes, either due to some feature values being unknown (a common occurrence in real data) or due to time restrictions. In these situations we can simply consider only the reachable decision nodes whose associated predictions are large. This should not reduce the classifier's accuracy significantly since the nodes which are ignored have little influence on the classification. The indices printed in the left side of the decision nodes in Figure 4.2 indicate the boosting iteration on which the nodes were added. In general, lower indices correspond to more influential nodes that were added earlier in the boosting process.

4.4 Experiments and results

To examine the practical performance of the ADTree algorithm described in Section 4.2 we ran experiments on a collection of 19 data sets taken from the UC Irvine repository [11]. We restricted our attention to binary classification problems only.

We compared the performance of four types of classifiers : single decision trees, voted combinations of decision trees, voted combinations of decision stumps and alternating decision trees. Single decision trees and voted combinations of decision trees were constructed using the C5.0 algorithm. For voted combinations of decision trees we restricted C5.0 to use 10 rounds of boosting. Voted combinations of decision stumps and alternating decision trees were constructed using the ADTree algorithm of Section 4.2. For the voted combinations of decision stumps ADTree was simply restricted to generate alternating decision trees with a single layer of decision nodes.

As can be seen in Figure 4.3, ADTree quickly overfits for smaller data sets such as the `cleve` data set. This occurs when ADTree is used to build either voted combinations

of decision stumps or full alternating decision trees. For this reason we had to carefully choose when to stop the boosting process. To choose the number of boosting rounds we did 10-fold cross validation within the training set in each of our experiments. That is, each training set was randomly subdivided into 10 subsets and each subset was used individually for validation purposes with the remaining 9 subsets used for training purposes. The results from this were then averaged. We chose as the stopping time the round for which the average error on the validation set was minimized. We then reran the algorithm on the whole training set and stopped it at the chosen iteration.

Because of this time-consuming method for choosing the stopping iteration we tested ADTree using 3 runs of 10-fold cross validation. That is, each data set was randomly subdivided into 10 subsets and each subset was used individually for test purposes with the remaining 9 subsets used for training purposes. This was repeated 3 times and all of the results from this were averaged. We tested C5.0 by averaging over 10 runs of 10 fold cross-validation.

Figure 4.3 shows two typical situations for the variation in training and test error as a function of boosting round for ADTree and AdaBoost decision stumps. For the `cleve` data set both algorithms reach their best performance after less then 10 boosting rounds. As the number of rounds increases the training error continues to decrease but the test error increases. To overcome this overfitting behavior, early stopping of the boosting algorithm is critical. The `kr-vs-kp` data set is much larger, which is probably the reason that there is almost no difference between the training and test errors. Here we see that ADTree reaches a very small error after about 50 rounds while the error of AdaBoost decision stumps remains large even after 200 iterations. This is a case in which the general structure of the alternating decision tree gives it an advantage.

Table 4.1 summarizes the average size of the generated classifier for each of the four

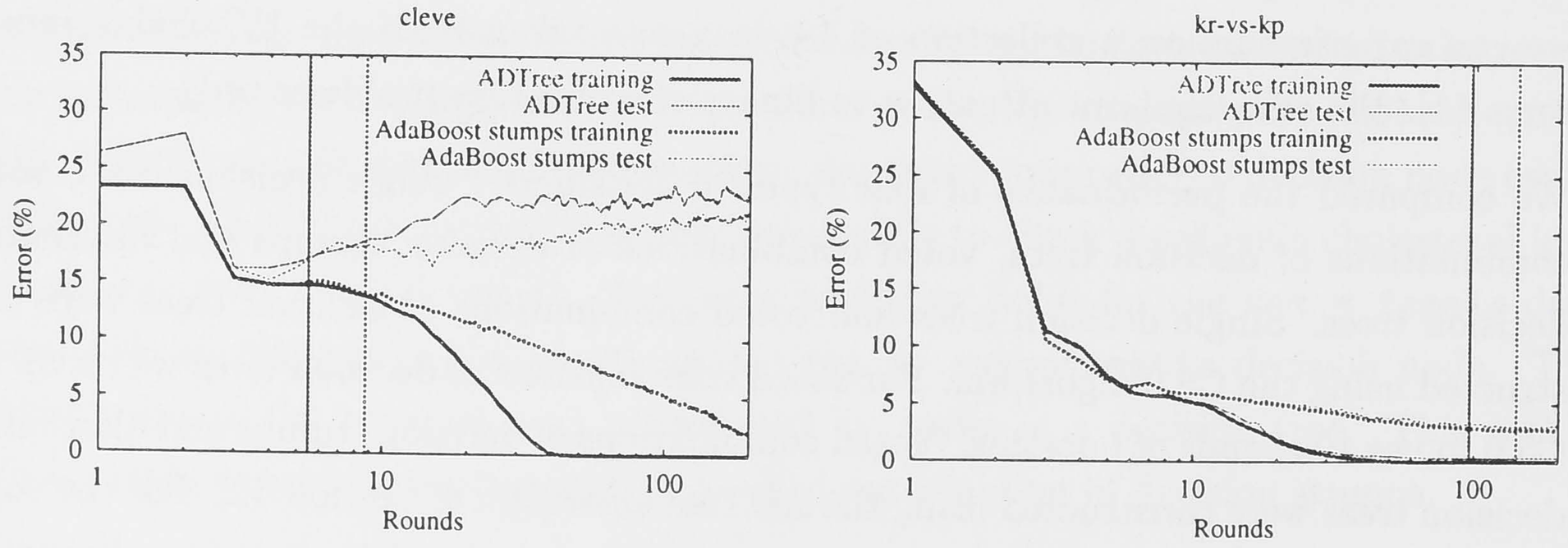


Figure 4.3: Average training and test error over 200 rounds of ADTree and AdaBoost decision stumps on the `cleve` and `kr-vs-kp` data sets. The average stopping points for ADTree and AdaBoost decision stumps are marked with solid vertical lines and dashed vertical lines respectively.

Data Set	Examples & Features	C5.0	C5.0 with boosting	AdaBoost decision stumps	ADTree
labor	57 : 16	4.3	61.0	27.6	43.3
promoters	106 : 57	16.6	134.6	133.8	309.5
hepatitis	155 : 20	9.8	197.8	47.9	42.5
sonar	208 : 60	14.3	197.7	463.1	338.1
cleve	303 : 13	27.2	446.8	9.2	5.5
ionosphere	351 : 34	13.8	216.8	51.3	299.3
house-votes-84	435 : 16	5.8	292.2	9.1	10.2
vote1	435 : 16	10.9	360.8	19.6	9.1
credit	690 : 15	18.3	770.2	31.1	160.3
breast-cancer	699 : 9	12.4	346.3	48.3	135.7
pima-indians	768 : 8	22.4	821.8	56.8	14.5
kkn	1000 : 16	7.0	50.1	10.1	5.3
hypo1	2514 : 29	6.9	235.4	33.0	23.0
hypothyroid	3163 : 25	6.6	567.9	58.7	6.5
sick-euthyroid	3163 : 25	13.3	1153.0	34.2	49.7
splice	3190 : 60	109.5	2637.0	94.8	318.8
kr-vs-kp	3198 : 36	29.2	849.4	141.4	96.3
dis	3772 : 29	14.2	629.4	129.0	118.4
agaricus-lepiota	8124 : 22	22.0	51.7	42.4	14.8

Table 4.1: Summary of average classifier sizes for C5.0, C5.0 with boosting, AdaBoost decision stumps and ADTree on all of the examined data sets.

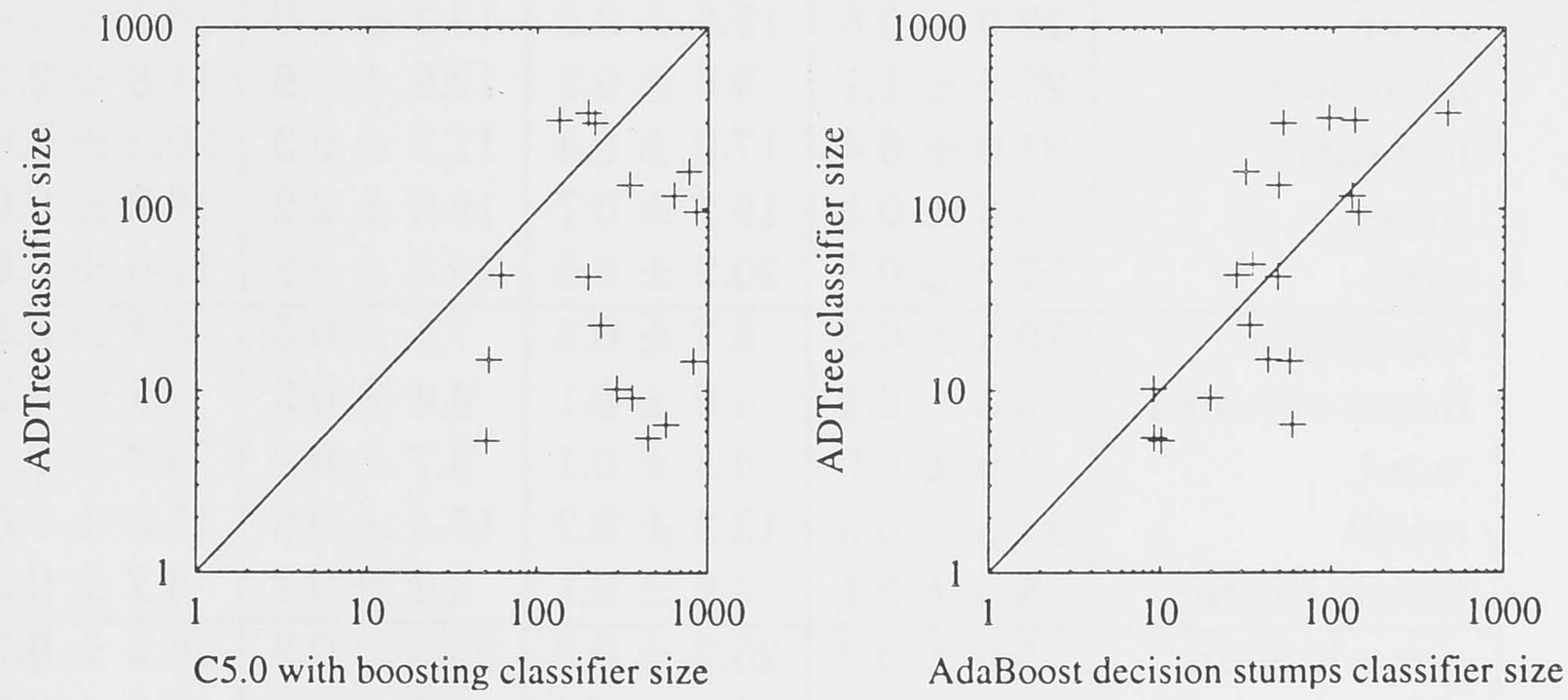


Figure 4.4: Comparisons of the average classifier sizes for C5.0 with boosting and AdaBoost decision stumps compared to ADTree on all of the examined data sets.

algorithms over the 19 data sets tested. For each data set the number of examples and features is listed along with the average size of the generated classifier for each algorithm tested. For C5.0 without boosting, the size is the average of the total number of decision nodes, while for C5.0 with boosting, the size is the average of the total number of decision nodes summed over the 10 voted decision trees. For AdaBoost decision stumps and

ADTree, the size is simply the average number of boosting rounds which is equal to the number of decision nodes.

Comparing the size of the classifiers we find that, in all but three cases, the classifiers generated by ADTree are much smaller then those generated by C5.0 with boosting. In fact, in more than half of the data sets tested, the classifier generated by ADTree is 5–10 times smaller than the classifier generated by C5.0 with boosting. A graphical comparison of the average classifier sizes for C5.0 with boosting, AdaBoost decision stumps and ADTree are summarized as two scatter plots in Figure 4.4.

Table 4.2 summarizes the average test error (with standard error) for each of the four algorithms over the 21 data sets tested. One interesting feature of these experimental results is that, in general, when C5.0 performed significantly better than AdaBoost decision stumps, so did ADTree and when AdaBoost decision stumps performed significantly better than C5.0, so did ADTree. This demonstrates the ability of alternating decision trees to represent both voted combinations of decision stumps and decision trees.

Surprisingly, in some of the data sets the best method was C5.0 without boosting. This is true for the `house-votes-84`, `hypo1`, `hypothyroid`, `sick-euthyroid` and `dis` data sets and for all of these data sets the tree size is small. We have no explanation for this behaviour.

Data Set	C5.0	C5.0 with boosting	AdaBoost decision stumps	ADTree
labor	19.9 ± 0.8	15.6 ± 0.8	13.2 ± 1.0	16.4 ± 3.4
promoters	22.4 ± 1.1	9.5 ± 0.8	12.9 ± 1.8	11.8 ± 2.1
hepatitis	21.0 ± 0.4	17.0 ± 0.4	17.3 ± 0.3	20.2 ± 0.6
sonar	25.6 ± 0.8	19.3 ± 0.7	18.5 ± 2.2	19.2 ± 0.6
cleve	27.2 ± 0.5	20.2 ± 0.5	16.5 ± 0.8	17.0 ± 0.6
ionosphere	10.3 ± 0.3	6.7 ± 0.2	7.1 ± 0.6	6.8 ± 0.3
house-votes-84	3.3 ± 0.1	4.0 ± 0.1	4.4 ± 0.3	3.7 ± 0.2
vote1	4.9 ± 0.2	4.3 ± 0.1	3.7 ± 0.3	4.3 ± 0.2
credit	14.3 ± 0.3	13.5 ± 0.2	15.4 ± 0.5	15.8 ± 0.0
breast-cancer	5.4 ± 0.1	3.9 ± 0.1	4.2 ± 0.7	4.7 ± 0.5
pima-indians	25.5 ± 0.2	25.3 ± 0.3	25.6 ± 0.2	26.6 ± 0.2
kkn	0.1 ± 0.0	0.1 ± 0.0	0.3 ± 0.1	0.3 ± 0.2
hypo1	0.2 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.4 ± 0.0
hypothyroid	0.7 ± 0.0	0.9 ± 0.0	1.1 ± 0.0	0.8 ± 0.0
sick-euthyroid	2.0 ± 0.0	2.3 ± 0.0	2.6 ± 0.1	2.4 ± 0.1
splice	4.2 ± 0.0	3.7 ± 0.0	6.0 ± 0.1	4.1 ± 0.1
kr-vs-kp	0.6 ± 0.0	0.4 ± 0.0	3.2 ± 0.0	0.4 ± 0.1
dis	1.0 ± 0.0	1.1 ± 0.0	1.6 ± 0.0	1.2 ± 0.1
agaricus-lepiota	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 4.2: Summary of test errors (with standard error) for C5.0, C5.0 with boosting, AdaBoost decision stumps and ADTree on all of the examined data sets.

For easier comparison, the average test errors for C5.0 with boosting, AdaBoost decision stumps and ADTree are summarized as two scatter plots in Figure 4.5. From these plots it can be seen that all three algorithms are competitive, with C5.0 with boosting having a slight advantage. In the larger data sets C5.0 with boosting and ADTree both outperform AdaBoost decision stumps. In the smaller data sets, AdaBoost decision stumps generally outperforms both C5.0 and ADTree, probably because it is less prone to overfitting.

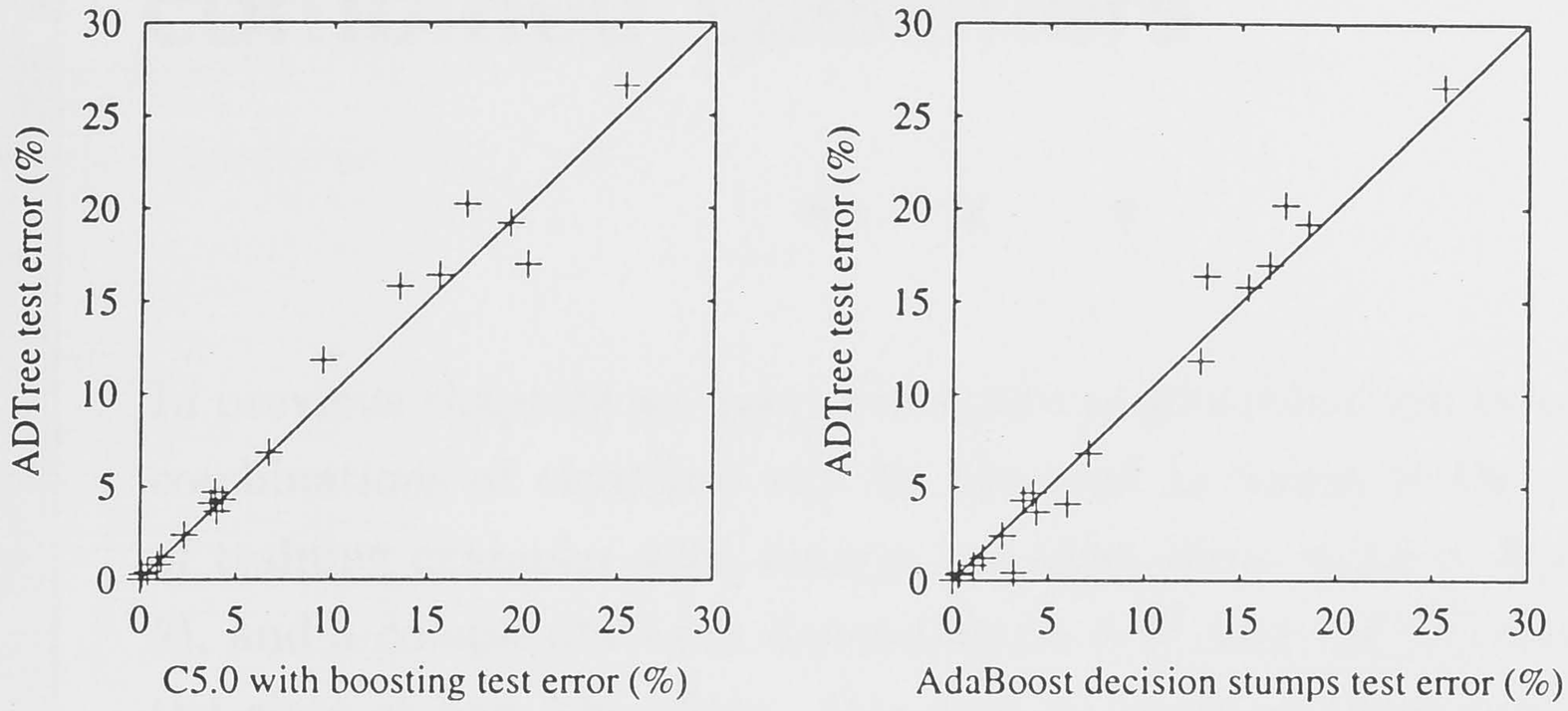


Figure 4.5: Comparisons of the average test errors for C5.0 with boosting and AdaBoost decision stumps compared to ADTree on all of the examined data sets.

CHAPTER

5 General margin bounds for combined classifiers

■ ■ ■

In previous chapters we have seen that the generalization error of voted combinations of classifiers can be bounded in terms of the proportion of training examples with margin less than some value θ, $\mathbf{P}_S(yF(x) \leq \theta)$, and a complexity term depending on $1/\theta^2$ and the VC dimension of the class of base classifiers. One way to think of these results is as a technique for adjusting the effective complexity of the class of combined classifiers by adjusting the parameter θ. Large values of θ correspond to low complexity and small values to high complexity. For a large value of θ, $\mathbf{P}_S(yF(x) \leq \theta)$ cannot be used to make fine distinctions between convex combinations in the class and consequently the effective complexity of the class is reduced. If we can find a convex combination in the class which has a small value of $\mathbf{P}_S(yF(x) \leq \theta)$ for a large value of θ, then the complexity term in the bound will be small and we will obtain a good bound on the generalization error of F. If we are forced to use a small value of θ in order to reduce $\mathbf{P}_S(yF(x) \leq \theta)$ then the complexity term will be large (as θ is small) and the VC bounds will give better results.

Since $\mathbf{P}_S(yF(x) \leq \theta) = \mathbf{E}_S 1(yF(x) \leq \theta)$, these bounds can be stated in terms of the sample average of $1(yF(x) \leq \theta)$, a threshold cost function of the training margins. From a practical standpoint, we can view these bounds as depending on a parameterized family of cost functions of the training margins. The family of cost functions is $\{1(yF(x) \leq \theta) : \theta \in (0, 1]\}$, parameterized by θ, and illustrated in Figure 5.1. To obtain the best possible bound for any particular convex combination F, we should choose the cost function by optimizing over choice of θ in order to minimize the bound. In this chapter we are concerned with upper bounds on the generalization error involving the sample average of more general cost functions of the training margins.

In Section 5.1, we give general conditions on parametrized families of cost functions that ensure that they can be used to give error bounds

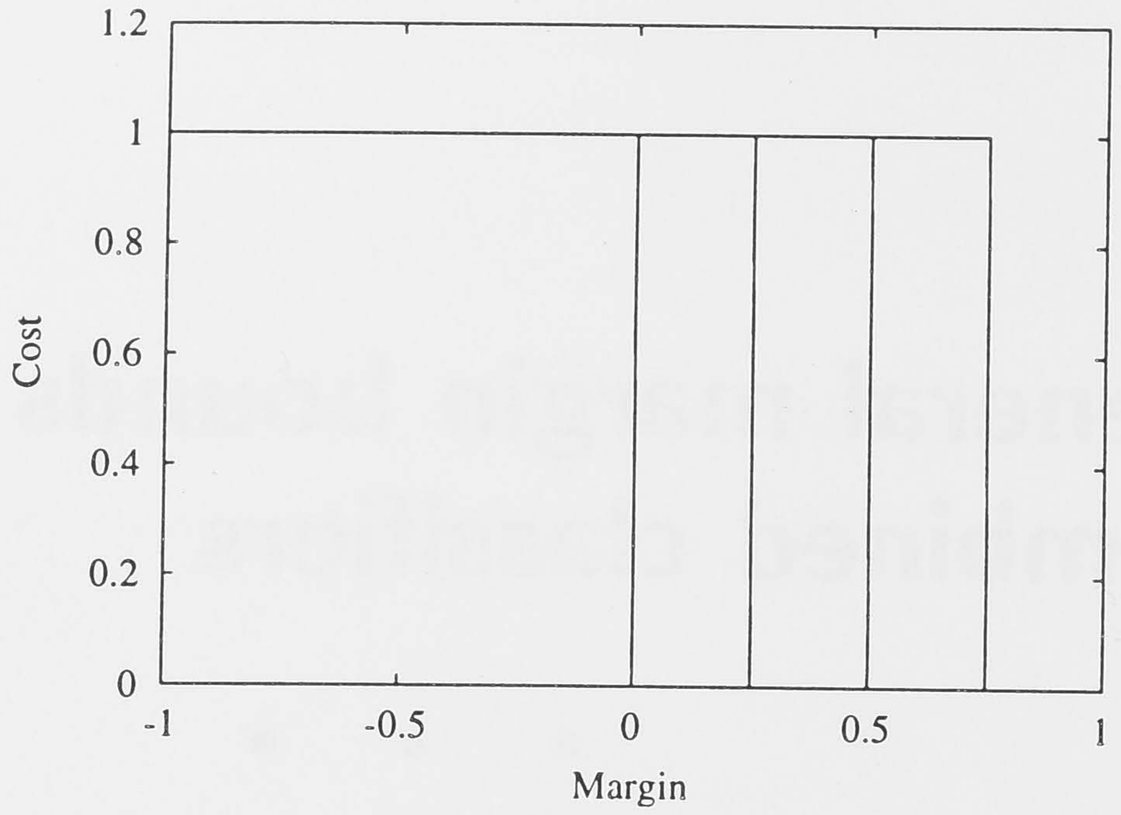


Figure 5.1: Cost functions $1(\alpha \leq \theta)$ (for $\theta = 0.25, 0.5$ and 0.75) compared to the function $1(\alpha \leq 0)$. Smaller values of θ correspond to closer approximations to $1(\alpha \leq 0)$.

for convex combinations of classifiers. These cost functions are all defined as the sample average of some function of the margin of an example. We then prove that, with high probability, the generalization error of a convex combination of classifiers is no more than the sample average of the cost function plus a complexity term involving a complexity parameter and the VC dimension of the class of base classifiers. The proof uses similar ideas to previous proofs (especially [60]), but is simpler.

Then, in Section 5.2, we examine a particular family of cost functions which give improved upper bounds compared to existing results for convex combinations of classifiers.

5.1 Main result

In this section, we derive upper bounds on the generalization error of a thresholded convex combination of classifiers, in terms of the sample average of certain functions of the margin, which we call *margin cost functions*. These are functions mapping from the interval $[-1, 1]$ to $\mathbb{R}^+$.

The general result derived in this section involves a family of margin cost functions, indexed by an integer-valued parameter N, which measures the resolution at which we examine the margins. A large value of N (corresponding to high resolution and high effective complexity of the convex combination) gives a margin cost function that is close to the threshold function $1(yF(x) \leq 0)$, which takes value 1 when the margin is no more than 0 and 0 otherwise. In this case, the sample average of the cost function is a close approximation of the training error. A small value of N (corresponding to low resolution and low effective complexity of the convex combination) gives a larger margin cost function. There is a trade-off between the effective complexity (as measured by N)

and how much larger the margin cost function is than the function $1(yF(x) \leq 0)$.

The following definition gives suitable conditions on the family of margin cost functions that ensure this trade-off is not violated. The particular form of this definition is not important; it arises from the proof technique that is used for the main theorem. In particular, the functions Ψ_N are only used in the analysis, and will not concern us later.

Definition 5.1
A family $\{C_N : N \in \mathbb{N}\}$ *of margin cost functions is* B-admissible *for* $B \geq 0$ *if for all* $N \in \mathbb{N}$ *there is a function* $\Psi_N : [-1,1] \to [-B,B]$ *that satisfies*

$$1(\alpha \leq 0) \leq \mathbf{E}_{Z \sim Q_{N,\alpha}} (\Psi_N(Z)) \leq C_N(\alpha)$$

for all $\alpha \in [-1,1]$, *where* $\mathbf{E}_{Z \sim Q_{N,\alpha}}(\cdot)$ *denotes the expectation when* Z *is chosen randomly as* $Z = (1/N)\sum_{i=1}^{N} Z_i$ *with* $Z_i \in \{\pm 1\}$ *and* $\mathbf{P}(Z_i = 1) = (1+\alpha)/2$.

In the following section we show that the family of threshold cost functions defined by $C_N(\alpha) = 1(\alpha \leq \theta) + 2e^{-N\theta^2/8}$ is an example of a suitable parametrized family, with N growing roughly as $1/\theta^2$. For this particular family the main result of this chapter reduces to Theorem 1.9. However, we shall see that better choices are possible giving upper bounds that improve on Theorem 1.9.

The following theorem is the main result of this chapter and provides, for any B-admissible family of margin cost functions, an upper bound on the generalization error of any convex combination of classifiers. For any choice of margin cost function $C_N(\alpha)$ in the family, the bound depends on the sample average of the chosen margin cost function and on a complexity term which increases with N.

Theorem 5.1
For any B-*admissible family* $\{C_N : N \in \mathbb{N}\}$ *of margin cost functions, there exists a constant* κ *such that with probability at least* $1-\delta$ *over the random choice of* S *according to* D^m, *every* $F \in \mathrm{co}(\mathcal{F})$ *and every* $N > 0$ *satisfies*

$$\mathbf{P}_D(yF(x) \leq 0) \leq \mathbf{E}_S C_N(yF(x)) + \kappa\sqrt{\frac{1}{m}\left(NB^2 \,\mathrm{VCdim}(\mathcal{F}) \ln m + \ln(N(N+1)) + \ln(1/\delta)\right)^{1/2}}.$$

Proof

We define an approximating class for $\mathrm{co}(\mathcal{F})$ by

$$\mathcal{G} = \left\{ \tfrac{1}{N}\sum_{i=1}^{N} g_i : g_i \in \mathcal{F} \right\}.$$

Given $F(x) = \sum_t w_t f_t(x)$ where $f_t \in \mathcal{F}$ we define a distribution over $\mathcal{G}$ via $\mathbf{P}(g_i = f_t) =$

w_t. The probability of randomly choosing a $G \in \mathcal{G}$ according to this distribution which satisfies a predicate A will be denoted by $\mathbf{P}_G(A)$ while the expected value over random choice of G will be denoted by $\mathbf{E}_G$. Let $Q_{N,\alpha}$ be the distribution of the average of N independent draws from $\{\pm 1\}$ with $\mathbf{P}(Z = 1) = (1 + \alpha)/2$, as in Definition 5.1. Then for any fixed x and y, the distribution of $yG(x)$ is $Q_{N,yF(x)}$ when G is chosen randomly according to the distribution defined above.

We are now going to bound $\mathbf{P}_D(yF(x) \leq 0)$ in terms of $\mathbf{E}_S C_N(yF(x))$ and a complexity term c. By suitable choice of c we will obtain the result.

$$
\begin{aligned}
&\mathbf{P}_{D^m}(\exists F \in \mathrm{co}(\mathcal{F}), N > 0 : \mathbf{P}_D(yF(x) \leq 0) > \mathbf{E}_S C_N(yF(x)) + \mathbf{E}_G c) \\
&\quad \leq \mathbf{P}_{D^m}(\exists F, N, : \mathbf{E}_D \mathbf{E}_{Z \sim Q_{N,yF(x)}}(\Psi_N(Z)) > \mathbf{E}_S C_N(yF(x)) + \mathbf{E}_G c) \\
&\qquad \text{[by the } B\text{-admissibility condition]} \\
&\quad = \mathbf{P}_{D^m}(\exists F, N, : \mathbf{E}_G \mathbf{E}_D \Psi_N(yG(x)) > \mathbf{E}_S C_N(yF(x)) + \mathbf{E}_G c) \\
&\quad \leq \mathbf{P}_{D^m}(\exists F, N, : \mathbf{E}_G \mathbf{E}_D \Psi_N(yG(x)) > \mathbf{E}_S \mathbf{E}_{Z \sim Q_{N,yF(x)}}(\Psi_N(Z)) + \mathbf{E}_G c) \\
&\qquad \text{[by the } B\text{-admissibility condition]} \\
&\quad = \mathbf{P}_{D^m}(\exists F, N, : \mathbf{E}_G \mathbf{E}_D \Psi_N(yG(x)) > \mathbf{E}_G \mathbf{E}_S \Psi_N(yG(x)) + \mathbf{E}_G c) \\
&\quad \leq \mathbf{P}_{D^m}(\exists N, G \in \mathcal{G} : \mathbf{E}_D \Psi_N(yG(x)) > \mathbf{E}_S \Psi_N(yG(x)) + c)
\end{aligned}
$$

This can be bounded above by

$$
\sum_{N=1}^{\infty} \mathbf{P}_{D^m}(\exists G \in \mathcal{G} : \mathbf{E}_D \Psi_N(yG(x)) > \mathbf{E}_S \Psi_N(yG(x)) + c) \tag{5.1}
$$

by application of the union bound over N.

We bound this last probability in (5.1) using Lemma 1.1 with $\Psi = \{\Psi_N(yG(x)) : G \in \mathcal{G}\}$. For this choice of Ψ, because Ψ_N is a fixed function, $\Pi_\Psi(m)$ is bounded above by the product of the growth functions of the N choices of functions from $\mathcal{F}$. Hence, (5.1) is bounded above by

$$
4 \sum_{N=1}^{\infty} \prod_{i=1}^{N} \Pi_{\mathcal{F}}(2m) e^{-mc^2/2B^2}. \tag{5.2}
$$

Now letting

$$
c = \sqrt{\frac{2B^2}{m}} \left(\ln(N(N+1)) + \ln 4 + \sum_{i=1}^{N} \ln \Pi_{\mathcal{F}}(2m) + \ln(1/\delta) \right)^{1/2}
$$

implies that (5.2) is equal to $\sum_{N=1}^{\infty} \frac{1}{N(N+1)} \delta = \delta$. By application of Jensen's inequality,

$$
\mathbf{E}_G c \leq \sqrt{\frac{2B^2}{m}} \left(\ln(N(N+1)) + \ln 4 + N \ln \Pi_{\mathcal{F}}(2m) + \ln(1/\delta) \right)^{1/2}.
$$

Finally, bounding the growth function of $\mathcal{F}$ via Sauer's lemma and simplifying gives the required result. □

5.2 Cost functions

As mentioned earlier, by appropriate choice of margin cost function we can derive Theorem 1.9 as a specific case of Theorem 5.1. Let us consider the margin cost function

$$C_N(\alpha) = 1(\alpha \leq \theta) + 2e^{-N\theta^2/8}.$$

It is straightforward to show that this family is B-admissible for $B = 2$ using

$$\Psi_N(Z) = 1(Z \leq \theta/2) + e^{-N\theta^2/8}. \tag{5.3}$$

The proof of B-admissibility essentially reduces to a proof of Lemma 2.3. This is identical to the requirement for the proof (from [60]) of Theorem 1.9. Theorem 1.9 is then directly obtained from Theorem 5.1 by using this B-admissible family and setting

$$N = \left\lceil \frac{4}{\theta_0^2} \ln \frac{m\theta^2}{\mathrm{VCdim}(\mathcal{F})} \right\rceil.$$

We can obtain an immediate improvement on the result of Theorem 1.9 by instead choosing

$$C_N(\alpha) = \mathbf{E}_{Z \sim Q_{N,\alpha}} \left(\Psi_N(Z) \right) \tag{5.4}$$

with $\Psi_N(Z)$ defined as in (5.3). With this motivation we now consider the family

$$C_N(\alpha) = \mathbf{E}_{Z \sim Q_{N,\alpha}} \left(1 \left(Z \leq \sqrt{(\ln N)/N} \right) \right). \tag{5.5}$$

This is essentially equivalent to using (5.4) with θ chosen roughly as $\sqrt{(\ln N)/N}$. We drop the exponential term because it is absorbed into the complexity term in the final result.

Figure 5.2 compares the threshold cost function $1(\alpha \leq 0)$ to the cost functions $C_N(\alpha)$ defined by (5.5). Notice that, for larger values of N, the cost functions C_N are closer to the threshold function $1(\alpha \leq 0)$, which measures training error. As N is decreased, the cost function $C_N(\alpha)$ moves away from the step function, and when we use it to compare functions in $\mathrm{co}(\mathcal{F})$, fine distinctions become more blurred.

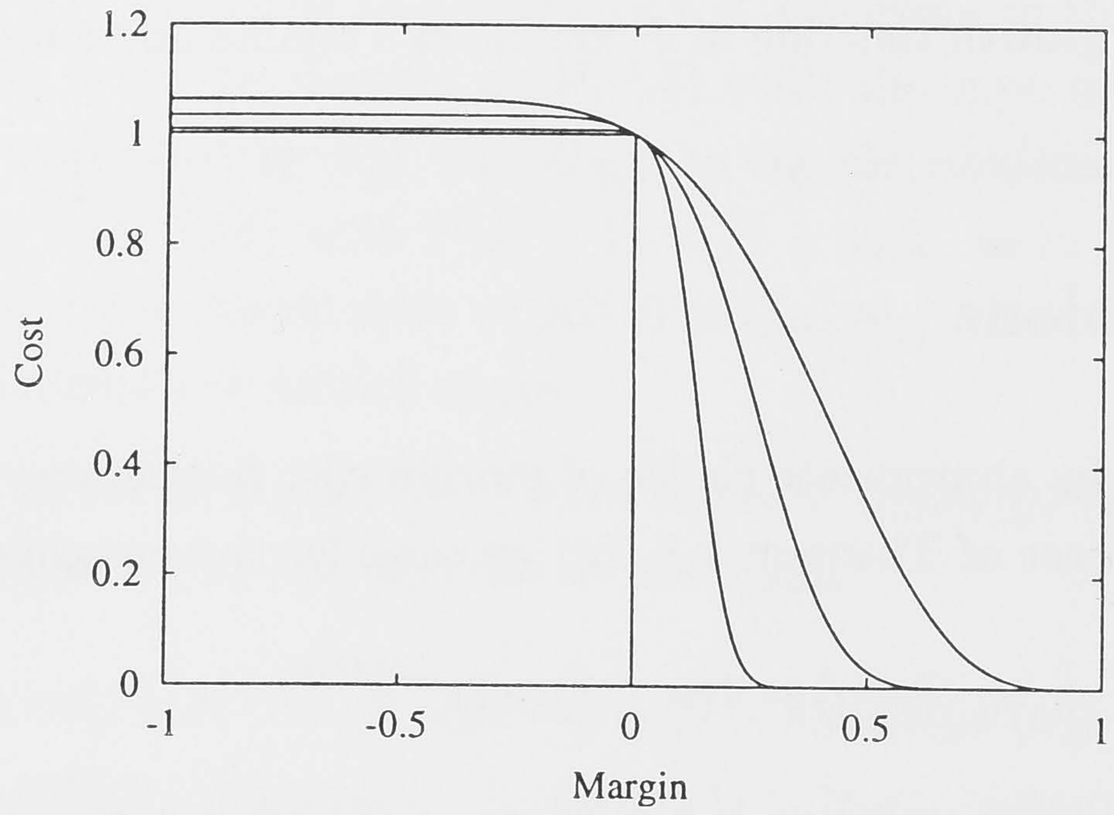


Figure 5.2: Cost functions $C_N(\alpha)$ (for $N = 15, 50$ and 200) compared to the function $1(\alpha \leq 0)$. Larger values of N correspond to closer approximations to $1(\alpha \leq 0)$.

5.3 Discussion

In this chapter we derived an upper bound on the generalization of convex combinations of classifiers which depends on the sample average of general cost functions of the margin and on a complexity term involving the complexity of the cost function and the VC dimension of the class of base classifiers.

In following chapters we will investigate the practical utility of these bounds by developing and evaluating new voting methods which attempt to produce voted classifiers which optimize the theoretically motivated cost functions of the margin suggested by the analysis of this chapter.

CHAPTER

6 DOOM - Direct optimization of margins

■ ■ ■

Theorems 1.9 and 2.2 give upper bounds on the generalization error of voted combinations of classifiers in terms of $\mathbf{P}_S(yF(x) \leq \theta)$, a threshold cost function of the training margins. Even though voting methods do not *explicitly* minimize $\mathbf{P}_S(yF(x) \leq \theta)$, Theorem 1.10 shows that AdaBoost *implicitly* minimizes such a cost function. Recall that Theorem 1.10 guarantees that $\mathbf{P}_S(yF(x) \leq \theta)$ decreases to zero exponentially quickly as boosting proceeds, provided that each base classifier generated by AdaBoost has weighted error smaller than $\frac{1}{2} - \theta - \gamma$, for some positive constant γ.

In Chapter 5 we presented an upper bound on the generalization error of voted combinations of classifiers in terms of more general cost functions of the training margins. Given that AdaBoost implicitly optimizes a cost function of the training margins and generalizes so well, it is natural to consider whether we can do better by explicitly optimizing some cost function of the margin. This is the main subject of the present chapter. In particular, we address the question: how useful are the theoretically motivated cost functions of Chapter 5 as error estimates?

In this chapter we investigate the problem of minimizing these cost functions by choosing an optimal set of classifier weights for an existing voted combination of classifiers. To overcome some severe computational difficulties, instead of considering the family described in Chapter 5, we consider a related family of piecewise linear cost functions. Section 6.1 describes *DOOM*, a gradient descent algorithm for this purpose. In Section 6.2, we describe experimental results with this algorithm on a selection of data sets from the UC Irvine repository. These results show that using DOOM to adjust the weights of a voted classifier generated using AdaBoost typically improves the generalization performance. Cumulative margin distribution plots show that in many cases the algorithm achieves these lower errors by sacrificing training error, in the interests of

reducing the new cost function. The margin distributions also suggest that the value of the minimum margin is not a critical factor in determining generalization performance.

6.1 Algorithm

We now consider how to select convex coefficients $w_1, \ldots, w_T$ for a sequence of binary classifiers $f_1, \ldots, f_T$ so that the combined classifier $\text{sign}(F(x))$, where $F(x) = \sum_{t=1}^{T} w_t f_t(x)$ has small generalization error. At this point we do not supply a procedure for also selecting the base classifiers. In all experiments we simply used the classifiers provided by AdaBoost, since the aim of the experiments was to investigate how useful are the error estimates provided by the cost functions of the previous chapter. In the next chapter, by considering voting methods within a more general framework, we develop procedures for choosing both the base classifiers and their weights in an iterative fashion.

If we take Theorem 5.1 at face value and ignore log terms, the best error bound is obtained if the weights $w_1, \ldots, w_T$ and the complexity parameter N are chosen to minimize

$$\frac{1}{m} \sum_{i=1}^{m} C_N(y_i F(x_i)) + \kappa \sqrt{\frac{N}{m}}, \tag{6.1}$$

where κ is a constant and $\{C_N\}$ is the family of cost functions shown in the left of Figure 6.1. However, there are two main difficulties with optimizing (6.1) directly. Firstly, the cost functions C_N are very flat for negative margins, which makes things difficult for local methods such as gradient descent. Secondly, although Theorem 5.1 provides an expression for the constant κ, in practical problems this will almost certainly be an overestimate and so our penalty for even moderately complex models will be too great.

To solve the first problem we bound C_N above by a monotone decreasing function and use this upper bound as our margin cost function. We considered the use of a sigmoid as a bounding cost function, but the existence of local minima at $\pm\infty$ caused difficulties for gradient descent approaches. Instead, we use the piecewise linear family of cost functions defined by

$$C_\theta(\alpha) = \begin{cases} (1.2 - \gamma) - \gamma\alpha & \text{for } -1 \leq \alpha \leq 0 \\ (1.2 - \gamma) - (1.2 - 2\gamma)\alpha/\theta & \text{for } 0 < \alpha \leq \theta \\ \gamma/(1-\theta) - \gamma\alpha/(1-\theta) & \text{for } \theta < \alpha \leq 1 \end{cases}$$

for $\theta \in (0, 1)$ (as shown in the right of Figure 6.1). Note that the parameter θ plays the role of the complexity parameter N in Theorem 5.1, only in this case smaller values of θ correspond to higher complexity classes. For all experiments, γ was fixed at 0.1.

To solve the second problem, instead of optimizing the average cost of the margins plus

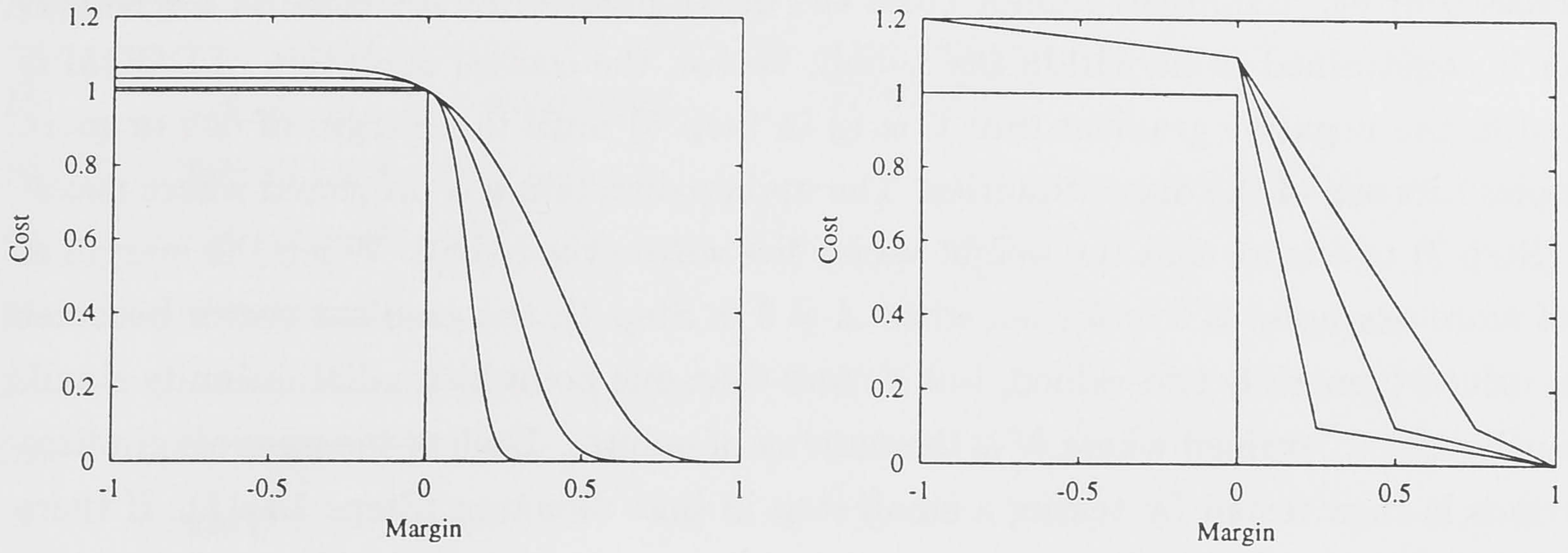


Figure 6.1: Theoretically motivated cost functions $C_N(\alpha)$ (for $N = 15, 50$ and 200) from Chapter 5 compared to piecewise linear upper bounds $C_\theta(\alpha)$ on the functions $C_N(\alpha)$. Larger values of N and smaller values of θ correspond to closer approximations to $1(\alpha \leq 0)$.

a penalty term over all values of the parameter θ, we instead estimate the optimal value of θ using an independent validation set. That is, for fixed values of θ in a discrete but fairly dense set we select weights minimizing

$$\frac{1}{m} \sum_{i=1}^{m} C_\theta(y_i f(x_i)) \tag{6.2}$$

and then choose the solution with smallest error on an independent validation set.

Unfortunately, even with the restriction to piecewise linear cost functions, the problem of optimizing (6.2) is still difficult. Fortunately, the nature of this cost function makes it possible to find successful heuristics (which is why we chose it).

The algorithm we have devised to optimize (6.2) is called *DOOM*, for *Direct Optimization Of Margins*. DOOM is basically a form of gradient descent, with two complications: we have to take account of the fact that our cost function is not differentiable at 0 and θ, and we have to ensure that the weight vector lies on the unit ball in l_1. Both these problems are addressed in DOOM, the pseudo-code of which is shown in Algorithm 6.1.

In order to help avoid problems with local minima we actually allow the weight vector to lie within the l_1-ball throughout optimization rather than on the l_1-ball. Since an increase in the l_1-norm of the weight vector generally corresponds to a decrease in the value of the cost function, the weight vector tends to naturally approach the surface of the l_1-ball as the optimization proceeds. If the weight vector reaches the surface of the l_1-ball and the update direction points out of the l_1-ball, it is simply projected back to the surface of the l_1-ball.

To understand the algorithm's operation, firstly observe that the gradient of (6.2) is a constant function of the weights $w = (w_1, \ldots, w_T)$ provided no example (x_i, y_i) 'crosses' one of the discontinuities at 0 or θ (i.e., provided the margin $y_i F(x_i)$ of an example does

not cross 0 or θ). Examples cannot cross the discontinuities at ± 1 because the weight vector is constrained to lie within the l_1-ball. Hence, the central operation of DOOM is to step in the negative gradient direction (g in Step 6) until the margin of one or more examples hits one of the discontinuities. The update direction g is projected where necessary (Step 7) to ensure that the weight vector lies within the l_1-ball. When the margin of one of more examples is 0 or θ (i.e., when $A \neq \emptyset$ in Step 4), the gradient vector becomes multi-valued (generally two-valued, but if more than one point hits a discontinuity simultaneously then 2^N-valued where N is the number of points). Each of the possible gradient directions is then tested by taking a small step in that direction (Steps 13–17). If there

---

Algorithm 6.1: DOOM

Require :

- A set of base classifiers $\{f_1, \ldots, f_T\}$ with each $f_t \colon X \to \{\pm 1\}$.
- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in X \times \{\pm 1\}$.
- A complexity parameter θ.
- Maximum number of active constraints M.
- $\epsilon = 2 \times$ machine precision.

1. **procedure** DOOM(w, E, `reset`)
2. let $F^w(x_i) := \sum_{t=1}^{T} w_t f_t(x_i)$ for $i = 1, \ldots, m$
3. let $\texttt{cost}(w) := \sum_{(x_i,y_i) \in S \setminus E} C_\theta(y_i F^w(x_i))$
4. let $A := \{(x_i, y_i) \in S \setminus E \colon y_i f_w(x_i) = 0 \text{ or } \theta\}$
5. **if** $A = \emptyset$ **then**
6. let $g := -\nabla_w \texttt{cost}(w)$
7. **call** PROJECT(w, g, E)
8. **else**
9. let $B := A$
10. **if** $|A| > M$ **then**
11. let B be a random subset of A of size M
12. let $N := |B|$
13. **for each** $(b_1, \ldots, b_N) \in \{\pm 1\}^N$
14. let $g^{(b_1,\ldots,b_N)} := -\nabla_w^{(b_1,\ldots,b_N)} \texttt{cost}(w)$, where $\nabla_w^{(b_1,\ldots,b_N)}$ means take the left or right derivative at the i^{th} example in B, according as b_i is $+1$ or -1
15. **call** PROJECT(w, g, E)
16. let $c^{(b_1,\ldots,b_N)} := \texttt{cost}(w + \epsilon g)$ - $\texttt{cost}(w)$
17. **end**

Continued overleaf ...

Algorithm 6.1 continued: DOOM

18. **let** $(b_1^*, \ldots, b_N^*) := \text{argmax}_{(b_1,\ldots,b_N)} \left\{c^{(b_1,\ldots,b_N)}\right\}$
19. **let** $c := c^{(b_1^*,\ldots,b_N^*)}$
20. **let** $g := g^{(b_1^*,\ldots,b_N^*)}$
21. **if** $c > 0$ **then**
22. **let** $E := E \cup A$
23. **call** PROJECT(w, g, E)
24. **end**
25. **end**
26. **if** $|g| < \epsilon$ **then**
27. **if** `reset` $=$ `FALSE` **then**
28. **call** DOOM$(w, \emptyset,$ `TRUE`$)$
29. **else**
30. **return**
31. **else**
32. **let** $\delta_1 := \text{argmin}_\delta \left\{\delta \colon \exists (x_i, y_i) \in S \setminus (E \cup A) \; : \; y_i F^{w+\delta g}(x_i) = 0 \text{ or } \theta\right\}$
33. **let** δ_2 be the distance to the closest face of the l_1-ball to w, in the direction g (ignoring those faces which already intersect w)
34. **call** DOOM$(w + \min(\delta_1, \delta_2)g, E,$ `FALSE`$)$
35. **end**
36.
37. **procedure** PROJECT(w, g, E)
38. Project g onto $\left\{v \in \mathbb{R}^T : \forall (x_i, y_i) \in E, \sum_{t=1}^{T} v_t h_t(x_i) = 0\right\}$
39. Project g onto all faces of the l_1-ball intersecting w

are too many gradient directions to be tested, a random subset of them are evaluated (Steps 10–11). The gradient direction corresponding to the maximum decrease in cost (Step 18) is then used as the next update direction g. If none of the directions tested lead to a decrease in the cost (Step 21), the examples whose margins lie on discontinuities of the cost function are added to a constraint set E (Step 22). In subsequent iterations the same stepping procedure above is followed except that the step direction is modified to ensure that the examples in E do not move (i.e., they remain on the discontinuity points of the cost function). This is achieved for a constraint set $E = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ by projecting any update direction onto the orthogonal subspace of the space spanned by the vectors $\{(f_1(x_1), \ldots, f_T(x_1)), \ldots, (f_1(x_k), \ldots, f_T(x_k))\}$ where $f_1, \ldots, f_T$ are the base classifiers. The procedure `PROJECT` performs this projection (Step 38) whilst also ensuring that the update direction lies along a face of the l_1-ball (Step 39). If no progress

is made in any iteration, the constraint set E is reset to zero. If still no progress is made the procedure terminates.

For each of the experiments reported in the next section, to increase the chance of finding the global minimum, the DOOM algorithm was called with 1000 random initial weight vectors and the solution with minimum cost was selected.

6.2 Experiments and results

For the experiments presented in this section we used a selection of data sets from the UC Irvine repository [11]. Only binary classification problems were considered, and due to the computational complexity of the DOOM algorithm we restricted our attention to the smaller UCI data sets. All of the experiments were repeated 10 times with 80%, 10% and 10% of the examples randomly selected for training, validation and test purposes respectively. The results were then averaged over the 10 repeats. For all of the experiments we used a base learner which exhaustively produces decision stumps. This ensured that the complexity of the base classifiers was constant.

Each experiment consisted of the following steps. Firstly, AdaBoost was run on the training data to produce a sequence of base classifiers and their corresponding weights. The number of classifiers in the combination was chosen by running AdaBoost for a large number of iterations and then choosing the combined classifier from the iteration corresponding to minimum error on the validation set. DOOM was then run on the classifiers produced by AdaBoost for a large range of θ values and 1000 random initial weight vectors for each value of θ. The weight vector (and θ value) with minimum error on the validation set was chosen as the final solution.

Figure 6.2 shows cumulative training margin distribution graphs for the `pima-indians`, `hepatitis`, `credit` and `cleve` data sets for both AdaBoost and DOOM (with optimal θ chosen using an independent validation set). For a given margin the value on the curve corresponds to the proportion of training examples with margin less than or equal to this value. The test errors for both algorithms are also shown for comparison, as short horizontal lines at margin 0.

There are several things worth noting about the margin distributions generated using DOOM as compared to those generated using AdaBoost. Firstly, they show that the value of the minimum training margin has no real impact on generalization performance. The idea of maximizing the minimum margin has been examined by Breiman [15] and Grove and Schuurmans [38]. Breiman presented several voting methods which provably maximize the minimum margin, while Grove and Schuurmans maximized the minimum margin explicitly by linear programming. In both cases a maximization of minimum margin at the expense of all other margins generally gave worse generalization perfor-

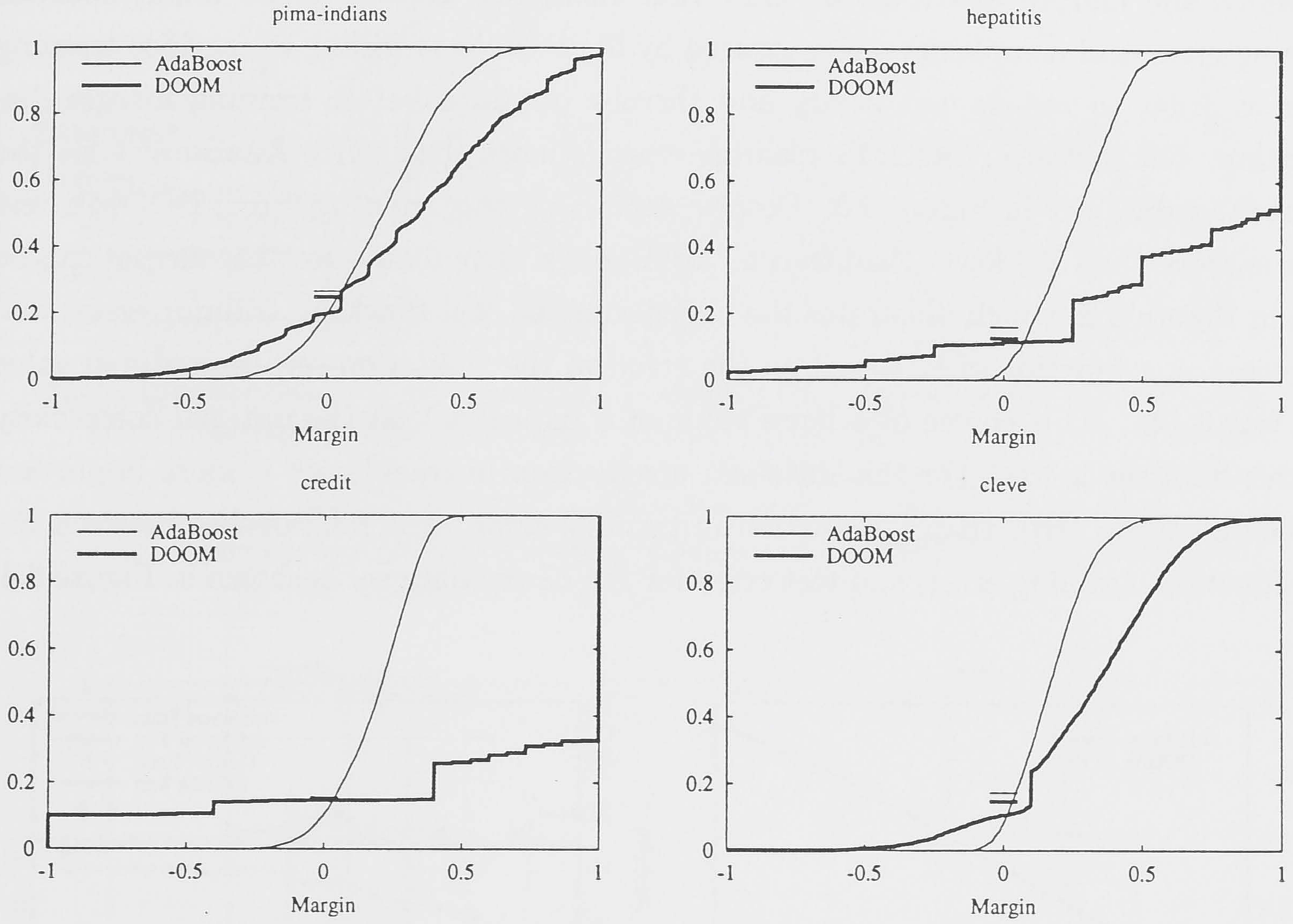


Figure 6.2: Cumulative training margin distributions for both AdaBoost and DOOM for the `pima-indians`, `hepatitis`, `credit` and `cleve` data sets. The test errors for AdaBoost and DOOM are marked as short horizontal lines at margin 0.

mance than AdaBoost. As can be seen in Figure 6.2, the generalization performance of the combined classifier produced by DOOM can be significantly better than that of the classifier produced by AdaBoost, despite having dramatically worse minimum training margin. This clearly demonstrates that the minimum margin is not a critical factor in determining generalization performance.

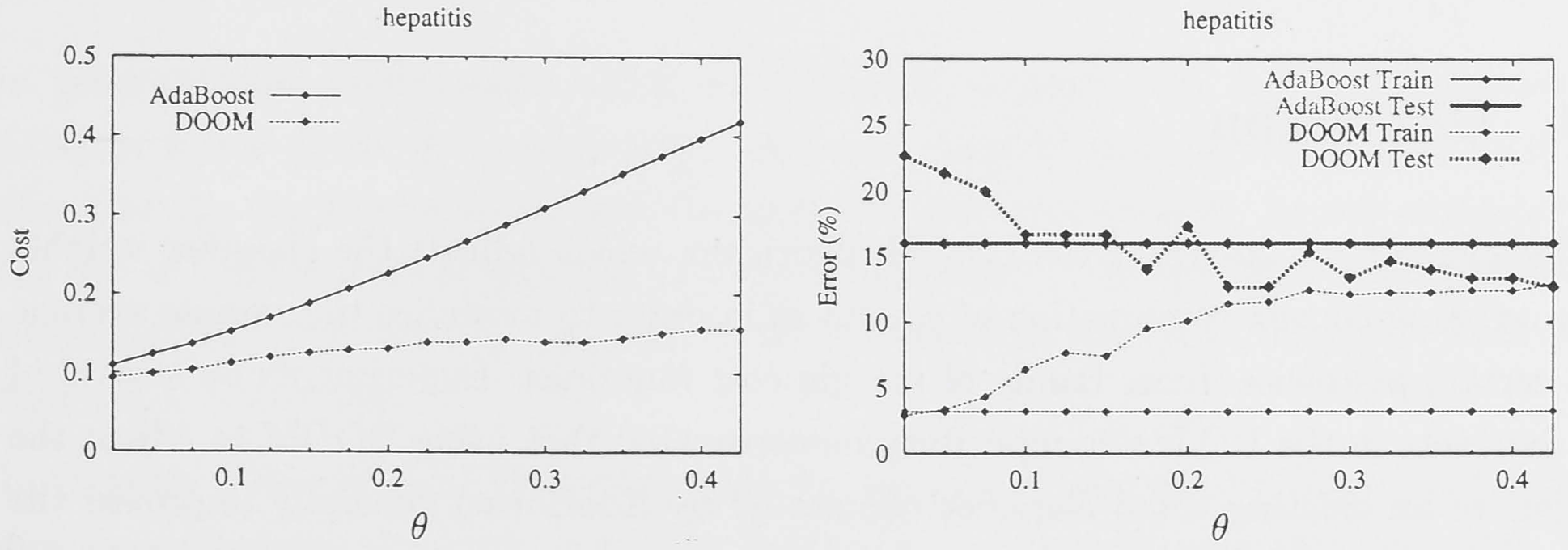


Figure 6.3: Plot of cost $\frac{1}{m}\sum_{i=1}^{m} C_\theta(y_i f(x_i))$, training and test error against θ for both AdaBoost and DOOM for the `hepatitis` data set.

Secondly, the margin distributions show that there is a balance to be found between training error and complexity (as measured by θ). DOOM is willing to sacrifice training error in order to reduce complexity and thereby obtain a better training margin distribution. For instance, DOOM's training error is more than twice AdaBoost's for the `hepatitis` data set in Figure 6.2. Despite sacrificing this training error, DOOM's test error is more than 3% lower than that of AdaBoost's. The reason for this success can be seen in Figure 6.3, which illustrates the changes in the cost function, training error, and test error as a function of θ. Based on the error on the validation set, the optimal value of θ was 0.425. This choice of a large value of θ indicates that the optimal complexity for this data set is low. For this data set, a reduction in complexity is more important to generalization error than a reduction in training error. The relationship between the cost function, training error, and test error for the `cleve` data set is shown in Figure 6.4.

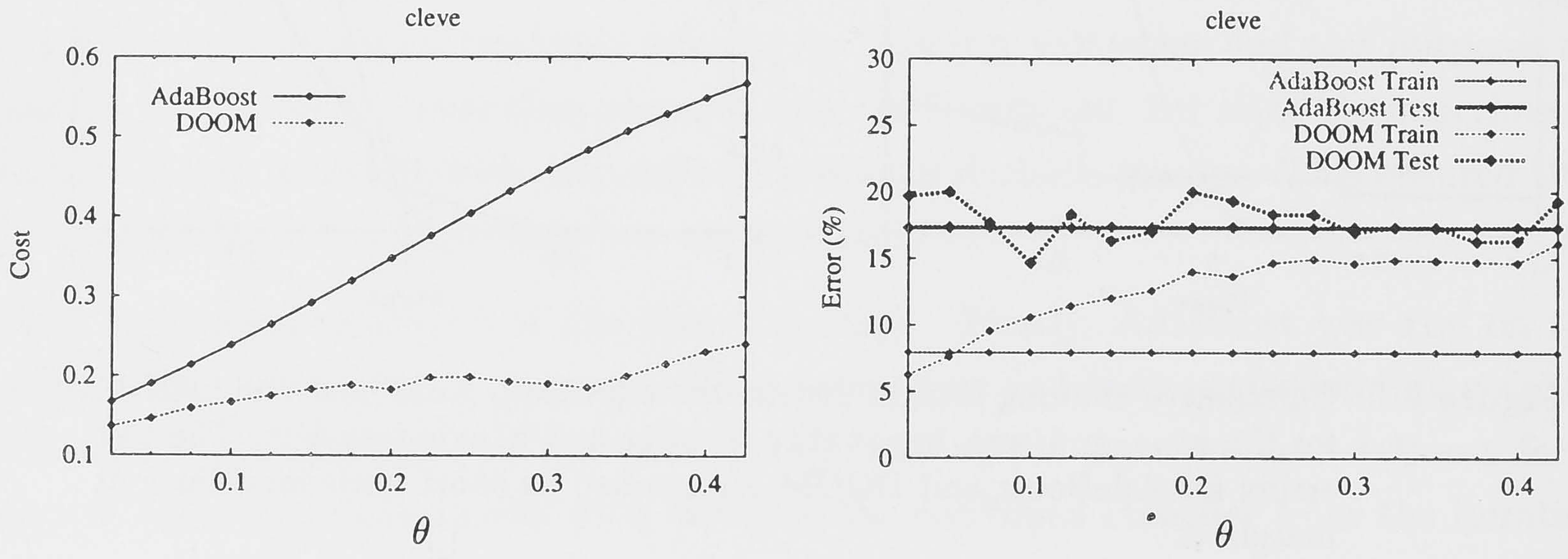


Figure 6.4: Plot of cost $\frac{1}{m}\sum_{i=1}^{m} C_\theta(y_i f(x_i))$, training and test error against θ for both AdaBoost and DOOM for the `cleve` data set.

A graphical representation of the difference in test error between AdaBoost and DOOM is shown in Figure 6.5. The improvement in test error exhibited by DOOM over AdaBoost (with standard error bars) is shown for each data set. These results show that DOOM generally outperforms AdaBoost.

6.3 Discussion

In this chapter we described the DOOM algorithm, which adjusts the classifier weights of an existing convex combination of classifiers in order to minimize the sample average of a certain piecewise linear family of margin cost functions. Experiments on several of the data sets in the UC Irvine repository demonstrated that using DOOM to adjust the weights of an existing voted classifier (generated by AdaBoost) generally improved the generalization performance.

The family of margin cost functions optimized by DOOM is qualitatively similar to the family suggested by the theoretical result of Chapter 5. The demonstrated improvement

	labor	promoters	hepatitis	sonar	cleve	ionosphere	vote1	credit	breast-cancer	pima-indians
Examples	57	106	155	208	303	351	435	690	699	768
Attributes	16	57	19	60	13	34	16	15	9	8
Stump	34.0	37.0	16.0	28.0	27.0	17.4	6.3	16.0	8.0	28.2
AdaBoost	**6.0**	19.0	16.0	17.5	17.3	**11.1**	**4.2**	15.7	4.6	26.4
DOOM	**6.0**	**17.0**	**12.7**	**15.0**	**14.7**	11.4	5.1	**15.1**	**3.8**	**24.7**

Table 6.1: Summary of test errors for a single decision stump, AdaBoost decision stumps and DOOM decision stumps on ten UCI data sets. The best test error for each data set is displayed in bold face. Note that since DOOM uses an independent validation set to choose the cost function parameter θ, we are comparing it to a version of AdaBoost modified to use an independent validation set for early stopping.

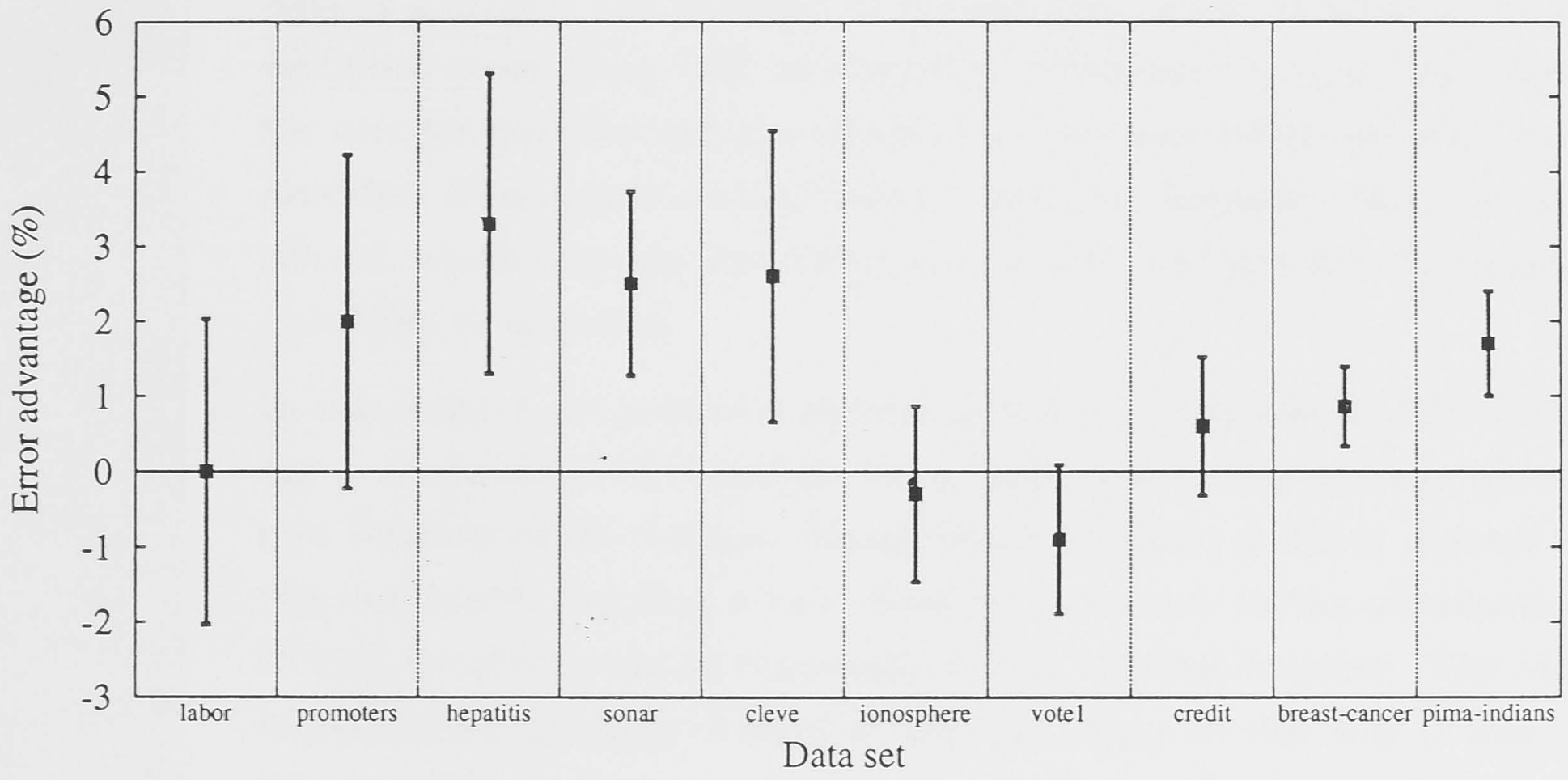


Figure 6.5: Summary of average test error advantage (with standard error bars) of DOOM over AdaBoost on ten UCI data sets. These results are averaged over 10 repeats.

in generalization performance which was obtained suggests that the cost functions of Chapter 5 are useful error estimates. However, DOOM only provides a method for choosing the classifier weights, not the base classifiers themselves. In our experiments the set of base classifiers was provided by AdaBoost. It seems likely that the set of base classifiers generated by AdaBoost will be suboptimal insofar as minimizing our cost function is concerned.

In Chapter 7 we will show that many existing voting methods can be viewed as gradient descent optimizers of margin cost functions within a suitable inner product space. As part of this new understanding of existing voting methods we will present a general algorithm for iteratively constructing a voted classifier (choosing both the base classifier

and its weight at each iteration) to minimize the sample average of any cost function of the training margin.

Then, in Chapter 8 we describe and evaluate a new algorithm, DOOM II, which is based on the general algorithm of Chapter 7. DOOM II iteratively constructs a voted combination of classifiers in order to minimize the sample average of a family of margin cost functions qualitatively similar to those suggested by the analysis of Chapter 5.

CHAPTER

7

A gradient descent view of voting methods

■ ■ ■

In Chapter 5 we presented improved upper bounds on the generalization error of a voted classifier in terms of the average over the training data of a certain cost function of the margins. Then, in Chapter 6, we described experiments with an algorithm which directly minimizes a similar cost function through the choice of weights associated with each base classifier. This algorithm exhibited performance improvements over AdaBoost, which suggests that these margin cost functions are appropriate quantities to optimize.

In this chapter, we present a general algorithm, MarginBoost, for choosing a combination of classifiers to optimize the sample average of any cost function of the margin. MarginBoost performs gradient descent in function space, choosing a base classifier to include in the combination at each iteration so as to maximally reduce the cost function. The idea of performing gradient descent in function space in this way is due to Breiman [15]. It turns out that, as in AdaBoost, the choice of the base classifier corresponds to a minimization problem involving weighted classification error. That is, for a certain weighting of the training data, the base classifier learning algorithm attempts to return a classifier that minimizes the weight of misclassified training examples.

There is a simpler and more abstract way to view the MarginBoost algorithm. In Section 7.1, we describe a class of algorithms (called AnyBoost) which are gradient descent algorithms for choosing linear combinations of elements of an inner product space so as to minimize some cost functional. Each component of the linear combination is chosen to maximize a certain inner product. (In MarginBoost, this inner product corresponds to the weighted training error of the base classifier.)

In Section 7.2, we show that this general class of algorithms includes as special cases a number of popular and successful voting methods, including Freund and Schapire's AdaBoost [35], Schapire and Singer's extension

of AdaBoost to combinations of real-valued functions [61], and Friedman, Hastie and Tibshirani's LogitBoost [37]. That is, all of these algorithms implicitly minimize some margin cost function by gradient descent.

In Section 7.3, we give convergence results for AnyBoost and its derivatives. For MarginBoost with a convex cost function, these results show that, with a particular choice of the step size, if the base classifier minimizes the appropriate weighted error then the algorithm converges to the global minimum of the cost function.

7.1 Optimizing cost functions of the margin

Although the abstract algorithms of the following section apply to many different machine learning settings, our primary interest in this chapter is the construction of voted combinations of classifiers of the form $\text{sign}(F(x))$, where

$$F(x) = \sum_{t=1}^{T} w_t f_t(x),$$

$f_t : X \to \{\pm 1\}$ are base classifiers from some fixed class $\mathcal{F}$ and $w_t \in \mathbb{R}$ are the classifier weights.

Given a training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ of m labelled examples generated according to D we wish to construct a voted combination of classifiers of the form described above so that the generalization error, $P_D(\text{sign}(F(x)) \neq y)$, is small. In this chapter we take the approach of finding voted classifiers which minimize the sample average of some cost function of the margin. That is, for a training set S we want to find F such that

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} C(y_i F(x_i)) \tag{7.1}$$

is minimized for some suitable cost function $C : \mathbb{R} \to \mathbb{R}$. Note that we are using the symbol C to denote both the cost function of the real margin $yF(x)$, and the cost *functional* of the function F. The intended interpretation should always be clear from the context.

This choice of optimization problem is motivated by the main theorem of Chapter 5 and the experimental results of Chapter 6. The main theorem of Chapter 5 provided an upper bound on the generalization error of convex combinations of classifiers in terms of the sample average of some cost function of the margin. The experimental results of Chapter 6 showed that generalization error typically improved after using the DOOM algorithm to modify the weights of an existing voted classifier so as to minimize the sample average of some cost function of the margin.

7.1.1 AnyBoost

One way to produce a weighted combination of classifiers which optimizes (7.1) is by gradient descent in function space, an idea first proposed by Breiman [15]. Here we present a more abstract treatment that shows how many existing voting methods may be viewed as gradient descent in a suitable *inner product* space.

At an abstract level we can view the base classifiers $f \in \mathcal{F}$ and their combinations F as elements of an inner product space $(\mathcal{X}, \langle \cdot, \cdot \rangle)$. In our case, $\mathcal{X}$ is a linear space of functions that contains $\text{lin}(\mathcal{F})$, the set of all linear combinations of functions in $\mathcal{F}$, and the inner product is defined by

$$\langle F, G \rangle := \frac{1}{m} \sum_{i=1}^{m} F(x_i) G(x_i) \tag{7.2}$$

for all $F, G \in \text{lin}(\mathcal{F})$. However, the AnyBoost algorithms defined in this section and their convergence properties studied in Section 7.3 are valid for any inner product. For example, they will hold in the case $\langle F, G \rangle := \int_X F(x) G(x) dP(x)$ where P is the marginal distribution on the input space generated by D.

Now suppose we have a function $F \in \text{lin}(\mathcal{F})$ and we wish to find a new $f \in \mathcal{F}$ to add to F so that the cost $C(F + \epsilon f)$ decreases, for some small value of ϵ. Viewed in function space terms, we are asking for the 'direction' f such that $C(F + \epsilon f)$ decreases most rapidly. Viewing the cost C as a *functional* on $\text{lin}(\mathcal{F})$, the desired direction is simply $-\nabla C(F)$, the negative of the functional derivative of C at F. Here, $\nabla C(F)$ is the unique function (up to trivial translations by $f \in \mathcal{X}$ such that $\|f\| = 0$) such that for any $f \in \mathcal{X}$

$$C(F + f) = C(F) + \langle \nabla C(F), f \rangle + o(\|f\|). \tag{7.3}$$

If we assume that C is differentiable everywhere then

$$\nabla C(F)(x) := \left. \frac{\partial C(F + \alpha 1_x)}{\partial \alpha} \right|_{\alpha = 0}, \tag{7.4}$$

where 1_x is the indicator function of x. Since we are restricted to choosing our new function f from $\mathcal{F}$, in general it will not be possible to choose $f = -\nabla C(F)$, so instead we search for an f with greatest inner product with $-\nabla C(F)$. That is, we should choose f to maximize

$$-\langle \nabla C(F), f \rangle .$$

This can be motivated by observing that (7.3) implies that, to first order in ϵ,

$$C(F + \epsilon f) = C(F) + \epsilon \langle \nabla C(F), f \rangle$$

and hence the greatest reduction in cost will occur for the f maximizing $-\langle \nabla C(F), f \rangle$.

The preceding discussion motivates Algorithm 7.1, an iterative algorithm for finding linear combinations F of base classifiers in $\mathcal{F}$ that minimize the cost $C(F)$. Note that we have allowed the base classifiers to take values in an arbitrary set Y, we have not restricted the form of the cost or the inner product, and we have not specified what the step sizes should be. Appropriate choices for these will be made when we apply the algorithm to more concrete situations. Note also that the algorithm terminates when $-\langle \nabla C(F_{t-1}), f_t \rangle \leq 0$. That is, when the base learner $\mathcal{L}$ returns a base classifier f_t which *no longer points in the downhill direction* of the cost function $C(F)$. Thus, the algorithm terminates when, to first order, a step in function space in the direction of the base classifier returned by $\mathcal{L}$ would increase the cost. This corresponds to reaching a local minimum of the cost functional.

At this point we note that from a gradient descent viewpoint, we should strictly be choosing f to maximize $-\langle \nabla C(F), f/\|f\| \rangle$ and then choosing an appropriate step size for f. However, for the inner product (7.2), the induced norm, $\|f\|^2 = \langle f, f \rangle$, is constant in the classification case. For the regression case (or more general inner products), the convergence proofs of Section 7.3 require that the induced norm is bounded if we are choosing f to maximize $-\langle \nabla C(F), f \rangle$.

Algorithm 7.1: AnyBoost

Require :

- An inner product space $(\mathcal{X}, \langle \cdot, \cdot \rangle)$ containing functions mapping from X to a set Y with bounded induced norm $\|f\|^2 = \langle f, f \rangle$.
- A class of base classifiers $\mathcal{F}$ where $\text{lin}(\mathcal{F}) \subseteq \mathcal{X}$.
- A differentiable cost functional $C\colon \text{lin}(\mathcal{F}) \to \mathbb{R}$.
- A base learner $\mathcal{L}(F)$ that accepts $F \in \text{lin}(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large value of $-\langle \nabla C(F), f \rangle$.

let $F_0(x) := 0$
for $t := 1$ **to** T
  let $f_t := \mathcal{L}(F_{t-1})$
  if $-\langle \nabla C(F_{t-1}), f_t \rangle \leq 0$ **then**
    return F_{t-1}
  Choose w_t
  let $F_t := F_{t-1} + w_t f_t$
return F_T

7.1.2 AnyBoost.L_1

The AnyBoost algorithm can return an arbitrary linear combination of elements of the class of base classifiers. Such flexibility has the potential to cause overfitting. Indeed, Theorem 5.1 provides guaranteed generalization performance for certain classes of cost functions, provided the algorithm returns elements of $\text{co}(\mathcal{F})$, that is, *convex* combinations of elements from the class of base classifiers. This consideration motivates Algorithm 7.2—AnyBoost.L_1—a normalized version of AnyBoost that only returns functions in the convex hull of the class of base classifiers $\mathcal{F}$. For convenience, we will assume that the class $\mathcal{F}$ contains the zero function, or equivalently, that $\text{co}(\mathcal{F})$ denotes the convex cone containing convex combinations of functions from $\mathcal{F}$ *and* the zero function.

Notice that the stopping criterion of AnyBoost.L_1 is $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$, rather than $-\langle \nabla C(F_{t-1}), f_t \rangle \leq 0$. To see why, notice that at every iteration F_t must lie in $\text{co}(\mathcal{F})$. Hence, in incorporating a new component f_t, we update F_{t-1} to $(1-\alpha)F_{t-1} + \alpha f_t$ for some $\alpha \in [0,1]$. Hence, $F_t = F_{t-1} + \alpha(f_t - F_{t-1})$ which corresponds to stepping in the direction corresponding to $f_t - F_{t-1}$. Geometrically, $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$ implies that the change $F_t - F_{t-1}$ associated with the addition of f_t is not within 90° of $-\nabla C(F_{t-1})$.

Algorithm 7.2: AnyBoost.L_1

Require :

- An inner product space $(\mathcal{X}, \langle \cdot, \cdot \rangle)$ containing functions mapping from X to a set Y with bounded induced norm $\|f\|^2 = \langle f, f \rangle$.
- A class of base classifiers $\mathcal{F}$ where $\text{lin}(\mathcal{F}) \subseteq \mathcal{X}$.
- A differentiable cost functional $C\colon\ \text{co}(\mathcal{F}) \to \mathbb{R}$.
- A base learner $\mathcal{L}(F)$ that accepts $F \in \text{co}(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large value of $-\langle \nabla C(F), f \rangle$.

let $F_0(x) := 0$

for $t := 1$ to T

  let $f_t := \mathcal{L}(F_{t-1})$

  if $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$ then

    return F_{t-1}

  Choose w_t

  let $F_t := \dfrac{F_{t-1} + w_t f_t}{\sum_{s=1}^{t} |w_s|}$

return F_T

7.1.3 AnyBoost.L_2

AnyBoost.L_1 enforces an L_1 constraint on the size of the combined classifiers returned by the algorithm. Although for certain classes of cost functionals we have theoretical guarantees on the generalization performance of such algorithms (see Chapter 5), from an aesthetic perspective an L_2 constraint is more natural in an inner product space setting. In particular, we could then ask our algorithm to perform gradient descent on a regularized cost functional of the form

$$C(F) + \lambda \|F\|^2,$$

where λ is a regularization parameter, without needing to refer to the individual weights in the combination F (contrast with AnyBoost.L_1).

With an L_2 rather than L_1 constraint, we also have the freedom to allow the base learner to return general linear combinations in $\text{lin}(\mathcal{F})$, instead of single classifiers in $\mathcal{F}$. While we could modify AnyBoost.L_1 so that the base learner was allowed to return general convex combinations in $\text{co}(\mathcal{F})$, the optimal direction in which to move for AnyBoost.L_1 is in fact always a *pure* direction in $\mathcal{F}$.. The following lemma shows that, provided $\mathcal{F}$ is negation closed (i.e., $f \in \mathcal{F} \implies -f \in \mathcal{F}$), no element of $\text{co}(\mathcal{F})$ will be closer to the negative gradient direction than the closest element of $\mathcal{F}$.

Lemma 7.1

If $\mathcal{F}$ is negation closed, at any iteration t of AnyBoost.L_1, for any $g \in \text{co}(\mathcal{F})$,

$$\sup_{f \in \mathcal{F}} - \langle \nabla C(F_{t-1}), f \rangle \geq - \langle \nabla C(F_{t-1}), g \rangle .$$

Proof

For any $g \in \text{co}(\mathcal{F})$ we can write $g = \sum_s v_s g_s$ for some $v_s > 0$ such that $\sum_s v_s = 1$. Hence,

$$\begin{aligned} - \langle \nabla C(F), g \rangle &= - \langle \nabla C(F), \textstyle\sum_s v_s g_s \rangle \\ &= - \sum_s v_s \langle \nabla C(F), g_s \rangle \\ &\leq - \sum_s v_s \inf_{f \in \mathcal{F}} \langle \nabla C(F), f \rangle \\ &= \sup_{f \in \mathcal{F}} - \langle \nabla C(F), f \rangle . \end{aligned}$$

□

So a base learner that produces combinations will be no more powerful than a base learner returning a single classifier in the L_1 case. This is not true for the L_2 case. In general, a linear combination $F \in \text{lin}(\mathcal{F})$ will be closer to the negative gradient direction than any single base classifier, hence stepping in the direction of F should lead to a greater reduction in the cost function, while still ensuring the overall classifier constructed is an

element of $\text{lin}(\mathcal{F})$.

A base learner $\mathcal{L}$ that accepts a direction G and attempts to choose an $f \in \mathcal{F}$ maximizing $\langle G, f \rangle$ can easily be converted to a base learner that attempts to choose an $H \in \text{lin}(\mathcal{F})$ maximizing $\langle G, H \rangle$ via a procedure we call *InnerBoost*; the details are given in Algorithm 7.3. The algorithm essentially works by iteratively combining classifiers from $\mathcal{F}$ in order to fit the residual difference between the current combination and the target direction G. The output of InnerBoost would then be substituted for the output of $\mathcal{L}$ in the AnyBoost algorithm.

Algorithm 7.3: InnerBoost

Require :

- An inner product space $(\mathcal{X}, \langle , \rangle)$ (with induced norm $\|F\|^2 := \langle F, F \rangle$) containing functions mapping from X to a set Y.
- A class of base classifiers $\mathcal{F}$ where $\text{lin}(\mathcal{F}) \subseteq \mathcal{X}$.
- A differentiable cost functional C: $\text{lin}(\mathcal{F}) \to \mathbb{R}$.
- A base learner $\mathcal{L}(G)$ that accepts a 'direction' $G \in \mathcal{X}$ and returns $f \in \mathcal{F}$ with a large value of $\langle G, f \rangle$.
- A starting function $F_t \in \text{lin}(\mathcal{F})$.

let $G_0 := -\nabla C(F_t)/\|\nabla C(F_t)\|$
let $H_0 := 0$
for $t' := 1$ to T'
  let $G_{t'} := G_0 - H_{t'-1}$
  let $h_{t'} := \mathcal{L}(G_{t'})$
  let $H_{t'} := \alpha H_{t'-1} + \beta h_{t'}$, with the constraints $\|H_{t'}\| = 1$ and $\langle H_{t'}, G_{t'} \rangle$ is maximal
  if $\beta = 0$ then
    return $H_{t'}$
return $H_{T'}$

7.1.4 AnyBoost and margin cost functionals

Since the main aim of this chapter is optimization of margin cost functionals, in this section we specialize the AnyBoost and AnyBoost.L_1 algorithms of the previous two sections by restricting our attention to the inner product (7.2), the cost (7.1), and $Y = \{\pm 1\}$. In this case,

$$\nabla C(F)(x) = \begin{cases} 0 & \text{if } x \neq x_i, i = 1 \dots m \\ \frac{1}{m} y_i C'(y_i F(x_i)) & \text{if } x = x_i, \end{cases}$$

where $C'(z)$ is the derivative of the margin cost function with respect to z. Hence,

$$-\langle \nabla C(F), f \rangle = -\frac{1}{m^2} \sum_{i=1}^{m} y_i f(x_i) C'(y_i F(x_i)).$$

Any sensible cost function of the margin will be monotonically decreasing and hence $-C'(y_i F(x_i))$ will always be positive. Dividing through by $-\frac{1}{m^2} \sum_{i=1}^{m} C'(y_i F(x_i))$, we see that finding an f maximizing $-\langle \nabla C(F), f \rangle$ is equivalent to finding an f minimizing

$$-\sum_{i=1}^{m} y_i f(x_i) \frac{C'(y_i F(x_i))}{\sum_{i=1}^{m} C'(y_i F(x_i))}. \tag{7.5}$$

Since $Y = \{\pm 1\}$, $y_i f(x_i)$ is either 1 if $f(x_i) = y_i$ or -1 if $f(x_i) \neq y_i$. Hence (7.5) can be rewritten as

$$\sum_{i: f(x_i) \neq y_i} D(i) - \sum_{i: f(x_i) = y_i} D(i) = 2 \sum_{i: f(x_i) \neq y_i} D(i) - 1,$$

where $D(1), \ldots, D(m)$ is the distribution on $\{1, \ldots, m\}$ defined by

$$D(i) := \frac{C'(y_i F(x_i))}{\sum_{i=1}^{m} C'(y_i F(x_i))}.$$

So finding an f maximizing $-\langle \nabla C(F), f \rangle$ is equivalent to finding f minimizing the weighted error

$$\sum_{i:\, f(x_i) \neq y_i} D(i).$$

Making the appropriate substitutions in AnyBoost yields Algorithm 7.4, MarginBoost. Similarly, making the appropriate substitutions in AnyBoost.L_1 yields Algorithm 7.5, MarginBoost.L_1.

7.2 Understanding existing voting methods

Many of the most successful voting methods are, for the appropriate choice of cost function and step size, specific cases of the AnyBoost algorithm described above (or its derivatives).

The interpretation of AdaBoost as an algorithm which performs a gradient descent optimization of the sample average of some cost function of the margins has been examined by several authors [15, 26, 31, 37, 56]. To see that the AdaBoost algorithm (as shown in Algorithm 1.1) is in fact MarginBoost using the cost function $C(\alpha) = e^{-\alpha}$ we need only verify that the distributions and stopping criteria are identical. From Algorithm 1.1, the

distribution D_{t+1} from AdaBoost can be written as

$$\frac{\prod_{s=1}^{t} e^{-y_i w_s f_s(x_i)}}{m \prod_{s=1}^{t} Z_s}. \tag{7.6}$$

Since D_{t+1} is a probability distribution then

$$m \prod_{s=1}^{t} Z_s = \sum_{i=1}^{m} \prod_{s=1}^{t} e^{-y_i w_s f_s(x_i)} \tag{7.7}$$

and clearly

$$\prod_{s=1}^{t} e^{-y_i w_s f_s(x_i)} = e^{-y_i F_t(x_i)}. \tag{7.8}$$

Substituting (7.7) and (7.8) into (7.6) gives the MarginBoost distribution for the cost function $C(\alpha) = e^{-\alpha}$. From Algorithm 1.1, the stopping criterion of AdaBoost is

$$\sum_{i:\, f_t(x_i) \neq y_i} D_t(i) \geq \frac{1}{2}.$$

Algorithm 7.4: MarginBoost

Require :

- A differentiable cost function $C\colon \mathbb{R} \to \mathbb{R}$.
- A class of base classifiers $\mathcal{F}$ containing functions $f\colon X \to \{\pm 1\}$.
- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in X \times \{\pm 1\}$.
- A base learner $\mathcal{L}(S, D)$ that accepts a training set S and a distribution D on the training set, and returns base classifiers $f \in \mathcal{F}$ with small weighted error $\sum_{i:\, f(x_i) \neq y_i} D(i)$.

let $F_0(x) := 0$

for $t := 1$ to T

  let $D_t(i) := \dfrac{C'\big(y_i F_{t-1}(x_i)\big)}{\sum_{i=1}^{m} C'\big(y_i F_{t-1}(x_i)\big)}$ for $i = 1, \ldots, m$

  let $f_t := \mathcal{L}(S, D_t)$

  if $\sum_{i=1}^{m} D_t(i) y_i f_t(x_i) \leq 0$ then

    return F_{t-1}

  Choose w_t

  let $F_t := F_{t-1} + w_t f_t$

return F_T

This is equivalent to

$$\sum_{i:\, f_t(x_i)=y_i} D_t(i) - \sum_{i:\, f_t(x_i)\neq y_i} D_t(i) \leq 0,$$

which is identical to the stopping criterion of MarginBoost.

Given that we have chosen f_t we wish to choose w_t to minimize

$$\sum_{i=1}^{m} C(y_i F_{t-1}(x_i) + y_i w_t f_t(x_i)).$$

Differentiating with respect to w_t, setting this to 0 and solving for w_t gives

$$w_t = \tfrac{1}{2} \ln \left(\frac{\sum_{i:f_t(x_i)=y_i} D_t(i)}{\sum_{i:f_t(x_i)\neq y_i} D_t(i)} \right).$$

This is exactly the setting of w_t used in the AdaBoost algorithm. So for this choice of cost function it is possible to find a closed form solution for the line search for optimal step size at each round. Hence, AdaBoost is performing gradient descent on the cost functional

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} e^{-y_i F(x_i)}$$

Algorithm 7.5: MarginBoost.L_1

Require :

- A differentiable cost function $C\colon \mathbb{R} \to \mathbb{R}$.
- A class of base classifiers $\mathcal{F}$ containing functions $f\colon X \to \{\pm 1\}$.
- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in X \times \{\pm 1\}$.
- A base learner $\mathcal{L}(S, D)$ that accepts a training set S and a distribution D on the training set, and returns base classifiers $f \in \mathcal{F}$ with small weighted error $\sum_{i:\, f(x_i)\neq y_i} D(i)$.

let $F_0(x) := 0$

for $t := 1$ **to** T

  let $D_t(i) := \dfrac{C'\big(y_i F_{t-1}(x_i)\big)}{\sum_{i=1}^{m} C'\big(y_i F_{t-1}(x_i)\big)}$ for $i = 1, \ldots, m$

  let $f_t := \mathcal{L}(S, D_t)$

  if $\sum_{i=1}^{m} D_t(i) y_i \left[f_t(x_i) - F_{t-1}(x_i)\right] \leq 0$ **then**

    return F_{t-1}

  Choose w_t

  let $F_t := \dfrac{F_{t-1} + w_t f_t}{\sum_{s=1}^{t} |w_s|}$

return F_T

---

with step size chosen by a line search.

Schapire and Singer [61] examine AdaBoost in the more general setting where classifiers can produce real values in $[-1, 1]$ indicating their confidence in $\{\pm 1\}$-valued classification. The general algorithm they present is essentially AnyBoost with the cost function $C(yF(x)) = e^{-yF(x)}$ and base classifiers $f : X \to [-1, 1]$. They also present a base learning algorithm for decision trees which directly optimizes the exponential cost function of the margin at each iteration. This variant of boosting does not reduce to a gradient descent optimization.

Breiman introduced the ARC-X4 algorithm [15] as an alternative to AdaBoost which uses an entirely different distribution over training examples. The combined classifier update in ARC-X4 is $F_t = (F_{t-1} + f_t)/t$, while the distribution over training examples is approximately (within an additive constant)

$$D_t(i) = \left(\sum_{s=1}^{t-1} (1 - y_i f_s(x_i))/2 \right)^4 / Z_t,$$

where Z_t is a normalization factor used to ensure that D_t is a probability distribution. We can rewrite this distribution as

$$\begin{aligned} D_t(i) &= \left((t-1) - y_i \sum_{s=1}^{t-1} f_s(x_i) \right)^4 / Z_t \\ &= \left(1 - y_i \sum_{s=1}^{t-1} \tfrac{1}{t-1} f_s(x_i) \right)^4 / Z_t \\ &= \left(1 - y_i F_{t-1}(x_i) \right)^4 / Z_t, \end{aligned}$$

where Z_t is appropriately redefined at each step to normalize the resulting distribution. This is exactly the same distribution assigned by MarginBoost for a cost function $C(\alpha) = (1 - \alpha)^5$. Thus, ARC-X4 is essentially MarginBoost.L_1 with the cost function $C(\alpha) = (1 - \alpha)^5$ and a constant step size of 1.

Friedman *et al.* [37] gave an interpretation of AdaBoost as an approximation to maximum likelihood. From this viewpoint they developed a more direct approximation, LogitBoost, which exhibits similar performance. LogitBoost is essentially AnyBoost with the cost function $C(\alpha) = \log_2(1 + e^{-2\alpha})$ and step size chosen via a single Newton-Raphson step.

In [46], Lee *et al.* describe an iterative algorithm for constructing convex combinations of basis functions to minimize a quadratic cost function. They use a constructive approximation result to prove the rate of convergence of this algorithm to the optimal convex combination. This algorithm is very similar to a special case of AnyBoost, with a quadratic cost function $C(\alpha) = (1 - \alpha)^2$ and step size decreasing at the rate $1/t$. The difference lies in the fact that, rather than adding the basis function corresponding to

Algorithm	Cost function	Step size
AdaBoost [34]	$e^{-yF(x)}$	Line search
ARC-X4 [13]	$(1-yF(x))^5$	1
Confidence-rated AdaBoost [61]	$e^{-yF(x)}$	Line search
LogitBoost [37]	$\ln(1+e^{-2yF(x)})$	Newton-Raphson
Constructive NN algorithm [46]	$(1-yF(x))^2$	$1/t$

Table 7.1: Summary of existing voting methods which are essentially identical to special cases of the AnyBoost algorithm and its derivatives.

the gradient direction at each round, the Lee *et al.* algorithm adds the basis function which actually minimizes the quadratic cost directly for a fixed step size.

Table 7.1 summarizes the cost function and step size choices for which AnyBoost and its derivatives reduce to algorithms which are essentially identical to existing voting methods.

7.3 Convergence results

In this section we prove convergence results for the abstract algorithms AnyBoost and AnyBoost.L_1, under quite weak conditions on the cost functional C. The prescriptions given for the step sizes w_t in these results are for convergence guarantees only: in practice they will almost always be smaller than necessary, hence fixed small steps or some form of line search should be used.

Throughout this section we are interested in the limiting behaviour of AnyBoost (and its derivatives) and thus assume that the algorithms do not terminate after some fixed number of iterations T (although the algorithms can terminate due to internal termination conditions).

7.3.1 Convergence of AnyBoost

The following theorem supplies a specific step size for AnyBoost and characterizes the limiting behaviour with this step size.

Theorem 7.2

Let $C\colon \operatorname{lin}(\mathcal{F}) \to \mathbb{R}$ be any lower bounded, Lipschitz differentiable cost functional (that is, there exists $L > 0$ such that $\|\nabla C(F) - \nabla C(F')\| \leq L\|F - F'\|$ for all $F, F' \in \operatorname{lin}(\mathcal{F})$). Let $F_1, F_2, \ldots$ be the sequence of combined classifiers generated by the AnyBoost algorithm,

using step sizes

$$w_t := -\frac{\langle \nabla C(F_{t-1}), f_t \rangle}{L\|f_t\|^2}. \tag{7.9}$$

Then AnyBoost either terminates at some iteration t with $-\langle \nabla C(F_{t-1}), f_t \rangle \leq 0$, or $C(F_t)$ converges to some finite value C^*, in which case

$$\lim_{t\to\infty} \langle \nabla C(F_{t-1}), f_t \rangle = 0.$$

Proof

First we need a general lemma.

Lemma 7.3

Let $(\mathcal{X}, \langle , \rangle)$ be an inner product space with squared norm $\|F\|^2 := \langle F, F \rangle$ and let $C\colon \mathcal{X} \to \mathbb{R}$ be a differentiable functional with $\|\nabla C(F) - \nabla C(F')\| \leq L\|F - F'\|$ for all $F, F' \in \mathcal{X}$. Then for any $w > 0$ and $F, G \in \mathcal{X}$,

$$C(F + wG) - C(F) \leq w \langle \nabla C(F), G \rangle + \frac{Lw^2}{2}\|G\|^2.$$

Proof

Define $g\colon \mathbb{R} \to \mathbb{R}$ by $g(w) := C(F + wG)$. Then $g'(w) = \langle \nabla C(F + wG), G \rangle$ and hence

$$\begin{aligned} |g'(w) - g'(0)| &= |\langle \nabla C(F + wG) - \nabla C(F), G \rangle| \\ &\leq \|\nabla C(F + wG) - \nabla C(F)\|\|G\| \quad \text{by Cauchy-Schwartz} \\ &\leq Lw\|G\|^2 \quad \text{by Lipschitz continuity of } \nabla C. \end{aligned}$$

Thus, for $w > 0$,

$$g'(w) \leq g'(0) + Lw\|G\|^2 = \langle \nabla C(F), G \rangle + Lw\|G\|^2$$

which implies

$$\begin{aligned} g(w) - g(0) &= \int_0^w g'(\alpha)\, d\alpha \\ &\leq \int_0^w \langle \nabla C(F), G \rangle + L\alpha\|G\|^2\, d\alpha \\ &= w \langle \nabla C(F), G \rangle + \frac{Lw^2}{2}\|G\|^2. \end{aligned}$$

Substituting $g(w) = C(F + wG)$ on the left hand side gives the result. □

Now we can write:

$$C(F_{t-1}) - C(F_t) = C(F_{t-1}) - C(F_{t-1} + w_t f_t)$$

$$\geq -w_t \langle \nabla C(F_{t-1}), f_t \rangle - \frac{L w_t^2 \|f_t\|^2}{2} \quad \text{by Lemma 7.3.}$$

If $\|f_t\| = 0$ then $\langle \nabla C(F_{t-1}), f_t \rangle = 0$ and AnyBoost will terminate. Otherwise, the greatest reduction occurs when the right hand side is maximized. That is, when

$$w_t = -\frac{\langle \nabla C(F_{t-1}), f_t \rangle}{L\|f_t\|^2},$$

which is the step size in the statement of the theorem. Thus, for our stated step size,

$$C(F_{t-1}) - C(F_t) \geq \frac{\langle \nabla C(F_{t-1}), f_t \rangle^2}{2L\|f_t\|^2}. \tag{7.10}$$

If $-\langle \nabla C(F_{t-1}), f_t \rangle \leq 0$ then AnyBoost terminates. Otherwise, since C is bounded below, $C(F_{t-1}) - C(F_t) \to 0$ which implies $\langle \nabla C(F_{t-1}), f_t \rangle / \|f_t\| \to 0$. Since our norm is bounded this implies that $\langle \nabla C(F_{t-1}), f_t \rangle \to 0$. □

The next theorem shows that if the base learner can always find the best base classifier $f_t \in \mathcal{F}$ on each round of AnyBoost, and if the cost functional C is convex, then any accumulation point F of the sequence $F_1, F_2, \ldots$ generated by AnyBoost with step sizes given by (7.9) is guaranteed to be a global minimum. It is convenient to assume that the class of base classifiers $\mathcal{F}$ is *negation closed*, which means $f \in \mathcal{F}$ implies $-f \in \mathcal{F}$. In this case, a function f_t that maximizes $-\langle \nabla C(F_{t-1}), f_t \rangle$ always satisfies $-\langle \nabla C(F_{t-1}), f_t \rangle \geq 0$. For ease of exposition, we have assumed that rather than terminating when $-\langle \nabla C(F_{t-1}), f_t \rangle = 0$, AnyBoost simply continues to return F_t for all subsequent iterations t.

Theorem 7.4

Let C: $\text{lin}(\mathcal{F}) \to \mathbb{R}$ *be a* convex *cost functional with the properties in Theorem 7.2, and let* $F_1, F_2, \ldots$ *be the sequence of combined classifiers generated by the AnyBoost algorithm with step sizes given by* (7.9). *Assume that the class of base classifiers* $\mathcal{F}$ *is negation closed and that on each round the AnyBoost algorithm finds a function* f_t *maximizing* $-\langle \nabla C(F_{t-1}), f_t \rangle$. *Then the sequence* $F_1, F_2, \ldots$ *satisfies*

$$\lim_{t\to\infty} \sup_{f\in\mathcal{F}} -\langle \nabla C(F_t), f \rangle = 0, \tag{7.11}$$

and any accumulation point F *of* F_t *satisfies*

$$C(F) = \inf_{G\in \text{lin}(\mathcal{F})} C(G). \tag{7.12}$$

Proof

Equation (7.11) follows immediately from Theorem 7.2. For the proof of (7.12) we need the following more general lemma:

Lemma 7.5

Let C be a differentiable convex cost functional on an inner product space $(\mathcal{X}, \langle , \rangle)$ with norm $\|F\|^2 = \langle F, F \rangle$. Let $\mathcal{M}$ be any linear subspace of $\mathcal{X}$ and let $\mathcal{M}^\perp$ denote the perpendicular subspace to $\mathcal{M}$ ($\mathcal{M}^\perp = \{G \in \mathcal{X} \colon \langle G, F \rangle = 0 \ \forall F \in \mathcal{M}\}$). If $F \in \mathcal{M}$ satisfies

$$\nabla C(F) \in \mathcal{M}^\perp$$

then

$$C(F) = \inf_{G \in \mathcal{M}} C(G).$$

Proof

Consider $G \in \mathcal{M}$. By the convexity of C, for all $0 \leq \epsilon \leq 1$,

$$C\big((1-\epsilon)F + \epsilon G\big) \leq \big((1-\epsilon)C(F) + \epsilon C(G)\big).$$

This can be rewritten as

$$C\big(F + \epsilon(G - F)\big) \leq C(F) + \epsilon\big(C(G) - C(F)\big),$$

which implies that, by definition of $\nabla C(F)$,

$$C(F) + \epsilon \langle G - F, \nabla C(F) \rangle + o(\|\epsilon(G-F)\|) \leq C(F) + \epsilon\big(C(G) - C(F)\big).$$

Dividing through by ϵ and taking the limit as $\epsilon \to 0$ yields,

$$\langle G - F, \nabla C(F) \rangle \leq C(G) - C(F).$$

Since $G - F \in \mathcal{M}$ and $\nabla C(F) \in \mathcal{M}^\perp$, this implies $C(G) \geq C(F)$. □

Now let F be an accumulation point of F_t. By Lipschitz continuity of $\nabla C(F)$ and (7.11),

$$\sup_{f \in \mathcal{F}} -\langle \nabla C(F), f \rangle = 0,$$

which by the negation closure of $\mathcal{F}$ implies $\langle \nabla C(F), f \rangle = 0$ for all $f \in \mathcal{F}$, hence $\nabla C(F) \in \mathrm{lin}(\mathcal{F})^\perp$. Thus $F \in \mathrm{lin}(\mathcal{F})$ and $\nabla C(F) \in \mathrm{lin}(\mathcal{F})^\perp$, which by Lemma 7.5 implies (7.12). □

7.3.2 Convergence of AnyBoost.L_1

The following theorem supplies a specific step size for AnyBoost.L_1 and characterizes the limiting behaviour under this step size regime.

Theorem 7.6

Let C be a cost function as in Theorem 7.2. Let $F_1, F_2, \ldots$ be the sequence of combined

classifiers generated by the AnyBoost.L_1 algorithm, using step sizes

$$w_t := \frac{-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle}{L\|f_t - F_{t-1}\|^2 + \langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle}. \tag{7.13}$$

Then AnyBoost.L_1 either terminates at some iteration t with $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$, or $C(F_t)$ converges to some finite value C^*, in which case

$$\lim_{t\to\infty} \langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle = 0.$$

Proof

Note that the step sizes w_t are always positive. In addition, if the w_t are such that $\sum_{s=1}^{t} w_s < 1$ for all t then $\lim_{t\to\infty} w_t = 0$ and clearly the second case above will apply. So without loss of generality assume $\sum_{s=1}^{t-1} w_s = 1$. Applying Lemma 7.3, we have:

$$\begin{aligned} C(F_{t-1}) - C(F_t) &= C(F_{t-1}) - C\left(\frac{F_{t-1} + w_t f_t}{1 + w_t}\right) \\ &= C(F_{t-1}) - C\left(F_{t-1} + \frac{w_t}{1 + w_t}(f_t - F_{t-1})\right) \\ &\geq -\frac{w_t}{1 + w_t}\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle - \frac{L}{2}\left[\frac{w_t}{1 + w_t}\right]^2 \|f_t - F_{t-1}\|^2. \end{aligned} \tag{7.14}$$

If $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$ then the algorithm terminates. Otherwise, the right hand side of (7.14) is maximized when

$$w_t = \frac{-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle}{L\|f_t - F_{t-1}\|^2 + \langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle}$$

which is the step size in the statement of the theorem. Thus, for our stated step size,

$$C(F_{t-1}) - C(F_t) \geq \frac{\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle^2}{2L\|f_t - F_{t-1}\|^2},$$

which by the lower-boundedness of C implies $\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle / \|f_t - F_{t-1}\| \to 0$. Since our norm is bounded this implies that $\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \to 0$. □

The next theorem shows that if the base learner can always find the best base classifier $f_t \in \mathcal{F}$ on each round of AnyBoost.L_1, and if the cost function C is convex, then AnyBoost.L_1 is guaranteed to converge to the global minimum of the cost. As with Theorem 7.4, we have assumed that rather than terminating when $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle \leq 0$, AnyBoost.$L_1$ simply continues to return F_t for all subsequent iterations t.

Theorem 7.7

Let C be a convex *cost function with the properties in Theorem 7.2, and let $F_1, F_2, \ldots$ be the sequence of combined classifiers generated by the AnyBoost.L_1 algorithm using the*

step sizes in (7.13). *Assume that the class of base classifiers* $\mathcal{F}$ *is negation closed and that on each round AnyBoost.*L_1 *finds a function* f_t *maximizing* $-\langle \nabla C(F_{t-1}), f_t - F_{t-1} \rangle$. *Then*

$$\lim_{t \to \infty} \sup_{f \in \mathcal{F}} -\langle \nabla C(F_t), f - F_t \rangle = 0, \tag{7.15}$$

and any accumulation point F *of the sequence* $F_1, F_2, \ldots$ *satisfies*

$$C(F) = \inf_{G \in \mathrm{co}(\mathcal{F})} C(G), \tag{7.16}$$

where $\mathrm{co}(\mathcal{F})$ *is the set of all convex combinations of classifiers from* $\mathcal{F}$.

Proof

Equation (7.15) follows immediately from Theorem 7.6. Now let F be an accumulation point of F_t. By (7.15) and continuity of $\nabla C(F)$, for all $f \in \mathcal{F}$,

$$\langle \nabla C(F), f - F \rangle = 0,$$

or equivalently $\langle \nabla C(F), f \rangle = \langle \nabla C(F), F \rangle$ for all $f \in \mathcal{F}$. Using the same argument as in the proof of Lemma 7.5, any $G \in \mathrm{co}(\mathcal{F})$ has

$$\langle G - F, \nabla C(F) \rangle \leq C(G) - C(F).$$

But because $\mathcal{F}$ is negation closed, we can write $G = \sum w_i f_i$ where all w_i are positive and $\sum w_i = 1$. Then

$$\langle G - F, \nabla C(F) \rangle = \sum w_i \langle f_i, \nabla C(F) \rangle - \langle F, \nabla C(F) \rangle = 0.$$

It follows that $C(G) \geq C(F)$. □

7.4 Discussion

In this chapter we presented an abstract gradient descent algorithm, AnyBoost, for generating linear combinations of elements of an inner product space which optimize a cost functional, and a related algorithm, AnyBoost.L_1, for generating convex combinations. For convex cost functionals, we gave prescriptions for the step sizes in these algorithms guaranteeing convergence to the linear or convex combination corresponding to the global minimum of the cost functional. For cost functionals which are nonconvex, like those motivated by the main result of Chapter 5, these results only guarantee convergence to a local minimum.

For cost functions depending only upon the *margins* of the classifier on the training

set, AnyBoost and AnyBoost.L_1 become MarginBoost and MarginBoost.L_1. We showed that many existing algorithms for combining classifiers can be viewed as special cases of MarginBoost.L_1; each algorithm differing only in its choice of margin cost function and step size. In particular, AdaBoost is MarginBoost.L_1 with $e^{-\alpha}$ as the cost function of the margin α, and with a step size equal to the one that would be found by a line search.

In the next chapter we examine the performance of a new algorithm, DOOM II, which combines the margin cost functions suggested by the theoretical result of Chapter 5 with a derivative of the MarginBoost.L_1 algorithm.

CHAPTER

8 DOOM II

In Chapter 5, we presented a result bounding the error of a convex combination of classifiers in terms of the sample average of certain cost functions of the margin. The cost functions suggested by this result are significantly different from the cost functions that are implicitly minimized by the voting methods described in Section 7.2.

In this chapter we describe a new algorithm, DOOM II, which combines the gradient descent optimization procedures of Chapter 7 with the cost functions suggested by the main theoretical result of Chapter 5. The details of the DOOM II algorithm are presented in Section 8.1, while in Section 8.2 we present experimental results showing that DOOM II typically outperforms AdaBoost, and that this is especially true with label noise. In addition, we show that the theoretically motivated cost functions provide good estimates of the error of AdaBoost, in the sense that they can be used to predict its overfitting behaviour.

8.1 Algorithm

Given the theoretical motivations described in Chapters 5 and 7 we propose a new algorithm, *DOOM II*; the details are given in Algorithm 8.1. DOOM II is essentially a derivative of MarginBoost.L_1 using the cost function $C(\alpha) = (1 - \tanh(\lambda\alpha))/2$, where λ is an adjustable parameter of the cost function. This family of cost functions (parameterized by λ) is qualitatively similar to the family $C_N(\alpha)$ (parameterized by N) which was suggested by the main theoretical result of Chapter 5. These families are shown in Figure 8.1. We do not use the functions $C_N(\alpha)$ directly as our cost functions because in practice they may cause difficulties for the gradient descent procedure as they are very flat for negative margins. Using the family of sigmoid cost functions alleviates this problem.

DOOM II can thus be viewed as procedure for gradient descent optimiza-

tion of

$$\frac{1}{m}\sum_{i=1}^{m}(1-\tanh(\lambda y_i F(x_i)))/2, \tag{8.1}$$

where F is restricted to be a convex combination of classifiers from some base class $\mathcal{F}$. Henceforth we will refer to (8.1) as the *normalized sigmoid cost function* (normalized because the weights are normalized so F is a convex combination).

Choosing a value of λ corresponds to choosing a value of the complexity parameter N in Theorem 5.1. It is a data dependent parameter which measures the resolution at which we examine the margins. A large value of λ corresponds to a high resolution and hence high effective complexity of the convex combination. Thus, choosing a large value of λ amounts to a belief that a high complexity classifier can be used without overfitting. Conversely, choosing a small value of λ corresponds to a belief that a high complexity classifier can avoid overfitting only if it has large margins.

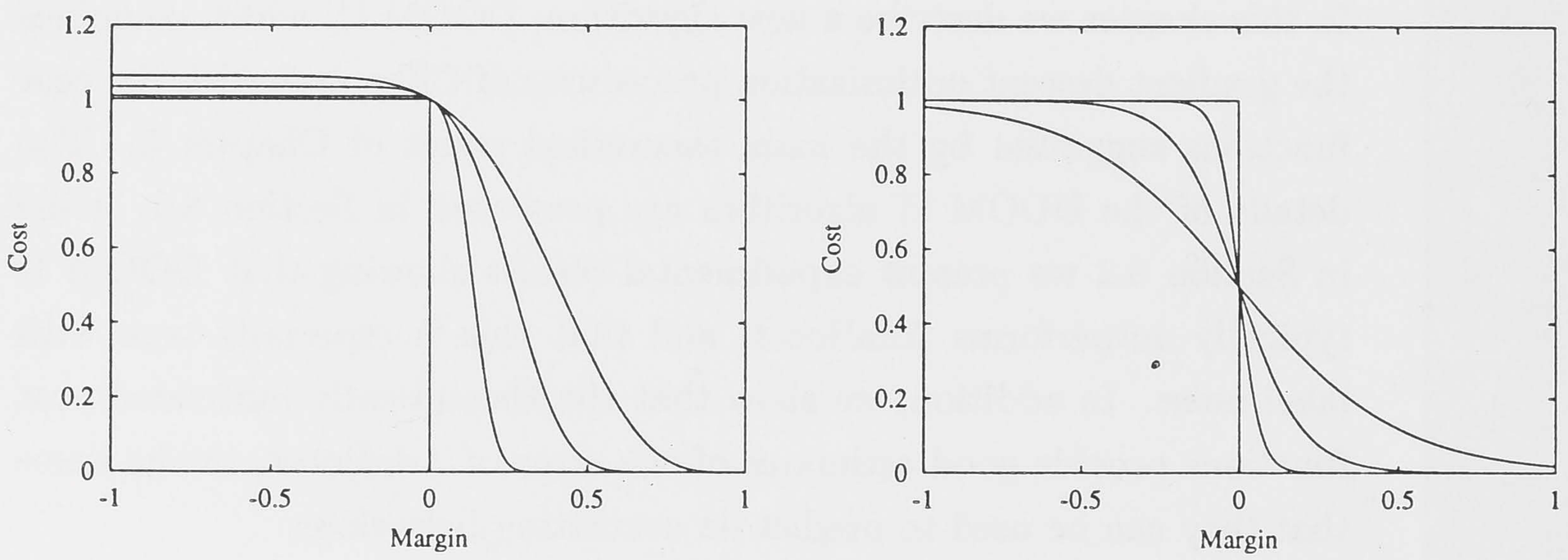


Figure 8.1: Theoretically motivated cost functions $C_N(\alpha)$ (for $N = 15, 50$ and 200) from Chapter 5 compared to the sigmoid cost functions $(1-\tanh(\lambda\alpha))/2$ (for $\lambda = 2, 5$ and 15) used in DOOM II. Larger values of N and λ correspond to closer approximations to $1(\alpha \leq 0)$.

AdaBoost had been perceived to be resistant to overfitting even though it can produce combinations involving very large numbers of classifiers. However, recent studies have shown that this is not the case, even for base classifiers as simple as decision stumps. Grove and Schuurmans [38] demonstrated that running AdaBoost for hundreds of thousands of rounds can lead to significant overfitting. Also, a number of authors [9, 22, 47, 57] showed that, by adding label noise, overfitting can be induced in AdaBoost even with relatively few classifiers in the combination.

Figure 8.2 compares the sigmoid cost function of DOOM II with the exponential cost function of AdaBoost and the logit cost function of LogitBoost. Notice that unlike the exponential and logit functions, the sigmoid function is nonconvex and for large negative margins is significantly smaller. The value of the sigmoid cost function will not increase significantly when an example with a large negative margin has its margin *decreased*.

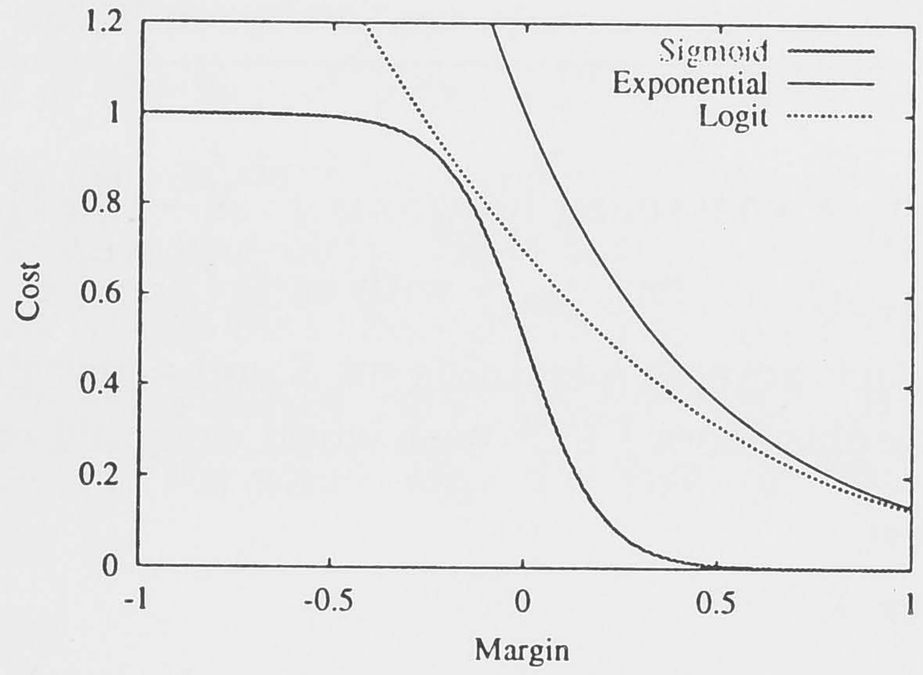


Figure 8.2: Comparison between the sigmoid, exponential and logit cost functions.

This allows DOOM II, unlike both AdaBoost and LogitBoost, to 'give up' on examples with large negative margins (and thus be more robust to label noise). This conclusion is supported by the experimental results presented in Section 8.2.

In the implementation of DOOM II described we are using a fixed small step size ϵ (for all of the experiments $\epsilon = 0.05$). In practice the use of a fixed ϵ could be replaced by a line search for the optimal step size at each round.

It is worth noting that since the l_1-norm of the classifier weights is fixed at 1 for each iteration and the cost function has the property that $C(-\alpha) = 1 - C(\alpha)$, the choice of λ is equivalent to choosing the l_1-norm of the weights while using the cost function $C(\alpha) = (1 - \tanh(\alpha))/2$.

Given that the normalized sigmoid cost function is nonconvex the DOOM II algorithm will suffer from problems with local minima. In fact, the following result shows that for cost functions satisfying $C(-\alpha) = 1 - C(\alpha)$, the MarginBoost.$L_1$ algorithm will strike a local minimum at the first step.

Lemma 8.1

Let $C\colon \mathbb{R} \to \mathbb{R}$ be any cost function satisfying $C(-\alpha) = 1 - C(\alpha)$. If MarginBoost.$L_1$ can find the optimal base classifier f_1 at the first time step, it will terminate at the next time step, returning f_1.

Proof

Let us assume without loss of generality that $C'(0) < 0$. With $F_0 = 0$, $\langle \nabla C(F_0), f \rangle = C'(0)\frac{1}{m}\sum_{i=1}^{m} y_i f(x_i)$ and so by assumption, f_1 will satisfy

$$\sum_{i=1}^{m} y_i f_1(x_i) = \inf_{f \in \mathcal{F}} \sum_{i=1}^{m} y_i f(x_i)$$

and $F_1 = f_1$. Now $C(-\alpha) = 1 - C(\alpha) \implies C'(-\alpha) = C'(\alpha)$, and since f_1 only takes

Algorithm 8.1: DOOM II

Require :

- A class of base classifiers $\mathcal{F}$ containing functions $f\colon X \to \{\pm 1\}$.
- A training set $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ with each $(x_i, y_i) \in X \times \{\pm 1\}$.
- A base learner $\mathcal{L}(S, D)$ that accepts a training set S and a distribution D on the training set, and returns base classifiers $f \in \mathcal{F}$ with small weighted error $\sum_{i:\ f(x_i) \neq y_i} D(i)$.
- A fixed small step size ϵ.
- A complexity parameter λ.

Let $F_0 := 0$

for $t := 1$ **to** T

Let $D_t(i) := \dfrac{1 - \tanh^2(\lambda y_i F_{t+1}(x_i))}{\sum_{i=1}^{m}(1 - \tanh^2(\lambda y_i F_{t+1}(x_i)))}$ for $i = 1, \ldots, m$

Let $f_t := \mathcal{L}(S, D_t)$

if $\sum_{i=1}^{m} D_t(i)[y_i f_t(x_i) - y_i F_{t-1}(x_i)] \leq 0$ **then**

return F_{t-1}

Let $w_t := \epsilon$

Let $F_t := \dfrac{F_{t-1} + w_t f_t}{\sum_{s=1}^{t} |w_s|}$

return F_T

values in $\{\pm 1\}$, we have for any f:

$$\langle \nabla C(F_1), f - F_1 \rangle = C'(1) \frac{1}{m} \sum_{i=1}^{m} y_i (f(x_i) - f_1(x_i)).$$

Thus, for all $f \in \mathcal{F}$, $-\langle \nabla C(F_1), f - F_1 \rangle \leq 0$ and hence MarginBoost.L_1 will terminate, returning f_1. □

A simple technique for avoiding this local minimum is to apply some notion of randomized initial conditions to the gradient descent procedure. Either the initial margins could be randomized or a random initial classifier could be chosen from $\mathcal{F}$. Initial experiments showed that both these techniques are somewhat successful, but could not guarantee avoidance of the single classifier local minimum unless many random initial conditions were tried (a computationally intensive prospect).

A more principled way of avoiding this local minimum is to remove f_1 from $\mathcal{F}$ after the first round and then continue the algorithm returning f_1 to $\mathcal{F}$ only when the cost goes below that of the first round. Since f_1 is a local minimum the cost is guaranteed to increase after the first round. However, if we continue to step in the best available

direction (the flattest uphill direction) we should eventually 'crest the hill' defined by the basin of attraction of the first classifier and then start to decrease the cost. Once the cost decreases below that of the first classifier we can safely return the first classifier to the class of available base classifiers. Of course, we have no guarantee that the cost will decrease below that of the first classifier at any round after the first. In practice, however, this is not a problem except for very small values of λ where the cost function is almost linear over $[-1, 1]$ (in which case the first classifier corresponds to a global minimum anyway).

8.2 Experiments and results

To compare the performance of DOOM II and AdaBoost a series of experiments were carried out on a selection of data sets taken from the UC Irvine repository [11]. We only considered binary classification problems. All of the experiments were repeated 100 times with 80%, 10% and 10% of the examples randomly selected for training, validation and test purposes respectively. The results were then averaged over the 100 repeats. For all of the experiments we used a base learner which exhaustively produces decision stumps. This fixed the complexity of the class of base classifiers and thus avoided any problems with the complexity of the combined classifier being dependent on the actual classifiers produced by the base learner.

For AdaBoost, the validation set was used to perform early stopping. AdaBoost was run for 2000 rounds and then the combined classifier was chosen from the round corresponding to minimum error on the validation set. For DOOM II, the validation set was used to set the data dependent complexity parameter λ. DOOM II was run for 2000 rounds with $\lambda = 2, 4, 6, 10, 15$ and 20 and the optimal λ was chosen to correspond to minimum error on the validation set after 2000 rounds. The typical behaviour of the test error as DOOM II proceeds is shown in Figure 8.3 for various values of λ. For small values of λ the test error converges to a value much worse than AdaBoost's test error. As λ is increased to the optimal value the test errors decrease. In the case of the `sonar` data set used in Figure 8.3 the test errors for AdaBoost and DOOM II with optimal λ are similar. Of course, with AdaBoost's adaptive step size it converges much faster than DOOM II (which uses a fixed step size).

AdaBoost and DOOM II were run on nine data sets to which varying levels of label noise had been applied. A summary of the experimental results is provided in Table 8.1. The attained test errors are shown for each data set for a single decision stump, AdaBoost applied to decision stumps and DOOM II applied to decision stumps with 0%, 5% and 15% label noise. A graphical representation of the difference in test error between AdaBoost and DOOM II is shown in Figure 8.4. The improvement in test error exhibited by DOOM II over AdaBoost (with standard error bars) is shown for each data set and

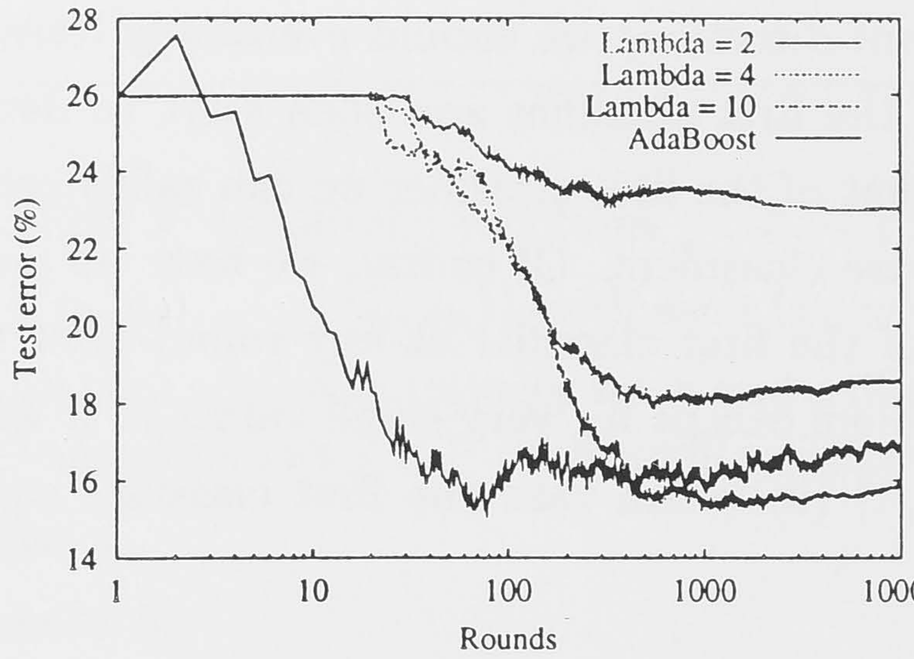


Figure 8.3: Test error for the `sonar` data set over 10000 rounds of AdaBoost and DOOM II with $\lambda = 2, 4$ and 10.

noise level. These results show that DOOM II generally outperforms AdaBoost and that the improvement is generally more pronounced in the presence of label noise.

		sonar	cleve	ionosphere	votel	credit	breast-cancer	pima-indians	hypo1	splice
Examples		208	303	351	435	690	699	768	2514	3190
Attributes		60	13	34	16	15	9	8	29	60
0% Label Noise	Stump	26.0	26.9	17.6	6.2	14.5	8.1	27.6	7.0	22.6
	AdaBoost	16.0	16.8	10.1	**3.5**	14.1	4.2	25.8	**0.5**	6.4
	DOOM II	**15.8**	**16.5**	**9.7**	4.5	**13.0**	**3.0**	**25.1**	0.7	**5.7**
5% Label Noise	Stump	30.4	29.0	21.7	10.6	18.0	12.1	29.7	12.4	26.4
	AdaBoost	**23.0**	21.6	16.7	9.6	17.5	9.0	**27.9**	8.6	13.9
	DOOM II	23.3	**20.3**	**14.6**	**9.4**	**17.0**	**8.0**	**27.9**	**7.1**	**12.1**
15% Label Noise	Stump	36.6	33.7	27.7	19.3	25.1	20.3	34.2	21.0	31.1
	AdaBoost	33.8	29.8	26.8	**19.0**	25.1	18.6	33.3	18.3	22.2
	DOOM II	**32.6**	**27.6**	**25.9**	**19.0**	**24.7**	**17.6**	**33.1**	**17.1**	**20.3**

Table 8.1: Summary of test errors for a single decision stump, AdaBoost decision stumps and DOOM II decision stumps with varying levels of label noise on nine UCI data sets. The best test error for each data set is displayed in bold face. Note that since DOOM II uses an independent validation set to choose the cost function parameter λ, we are comparing it to a version of AdaBoost modified to use an independent validation set for early stopping.

The effect of using the normalized sigmoid cost function rather than the exponential cost function is best illustrated by comparing the cumulative margin distributions generated by AdaBoost and DOOM II. Figure 8.5 shows comparisons for two data sets with 0% and 15% label noise applied. For a given margin, the value on the curve corresponds to the proportion of training examples with margin less than or equal to this value. These curves show that in trying to increase the margins of negative examples AdaBoost is willing to sacrifice the margin of positive examples significantly. In contrast, DOOM II

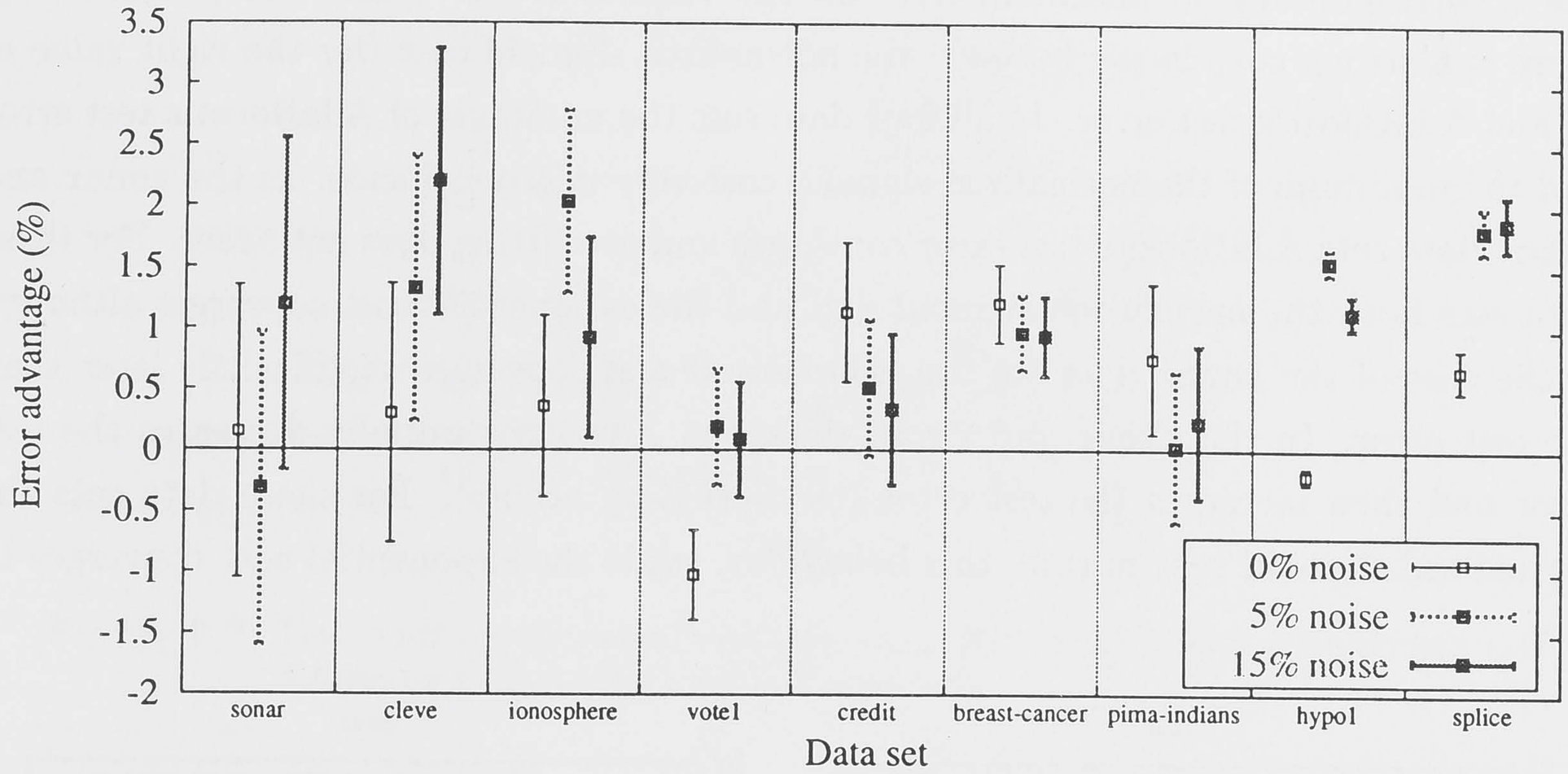


Figure 8.4: Summary of average test error advantage (with standard error bars) of DOOM II over AdaBoost with varying levels of noise on nine UCI data sets. These results are averaged over 100 repeats.

'gives up' on examples which have a large negative margin in order to reduce the value of the cost function.

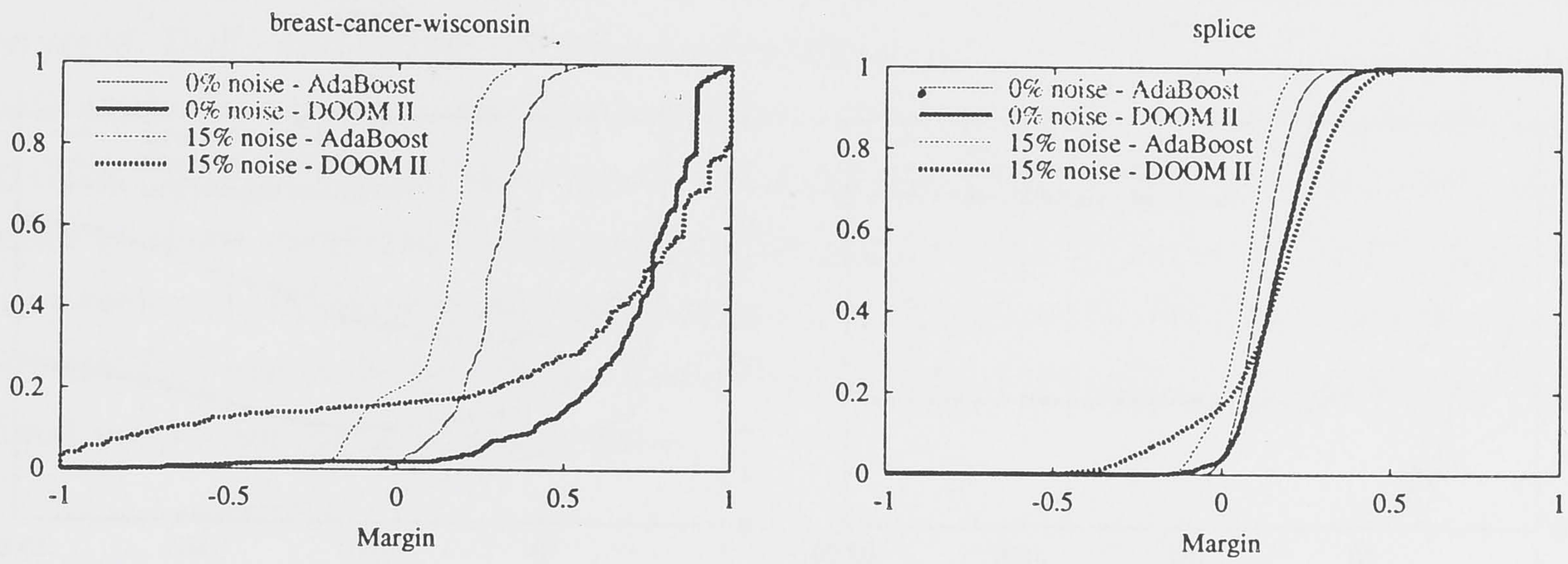


Figure 8.5: Cumulative training margin distributions for both AdaBoost and DOOM II with 0% and 15% label noise for the `breast-cancer` and `splice` data sets.

Given that AdaBoost suffers from overfitting and minimizes an exponential cost function of the margins, this cost function certainly does not relate to test error. How does the value of our proposed cost function correlate with AdaBoost's test error? The theoretical bound suggests that for the 'right' value of the data dependent complexity parameter λ our cost function and the test error should be closely correlated. Figure 8.6 shows the variation in the normalized sigmoid cost function, the exponential cost function and the test error for AdaBoost for four UCI data sets over 10000 rounds. As before, the values of these curves were averaged over 100 random train/validation/test splits. The value of λ used in each case was chosen by running DOOM II for various values of λ and choosing

the λ corresponding to minimum error on the validation set. These curves show that there is a strong correlation between the normalized sigmoid cost (for the right value of λ) and AdaBoost's test error. In all four data sets the minimum of AdaBoost's test error and the minimum of the normalized sigmoid cost very nearly coincide. In the `sonar` and `labor` data sets AdaBoost's test error converges and overfitting does not occur. For these data sets both the normalized sigmoid cost and the exponential cost converge, although in the case of the `sonar` data set the exponential cost converges significantly later than the test error. In the `cleve` and `vote1` data sets AdaBoost initially decreases the test error and then increases the test error (as overfitting occurs). For these data sets the normalized sigmoid cost mirrors this behaviour, while the exponential cost converges to zero.

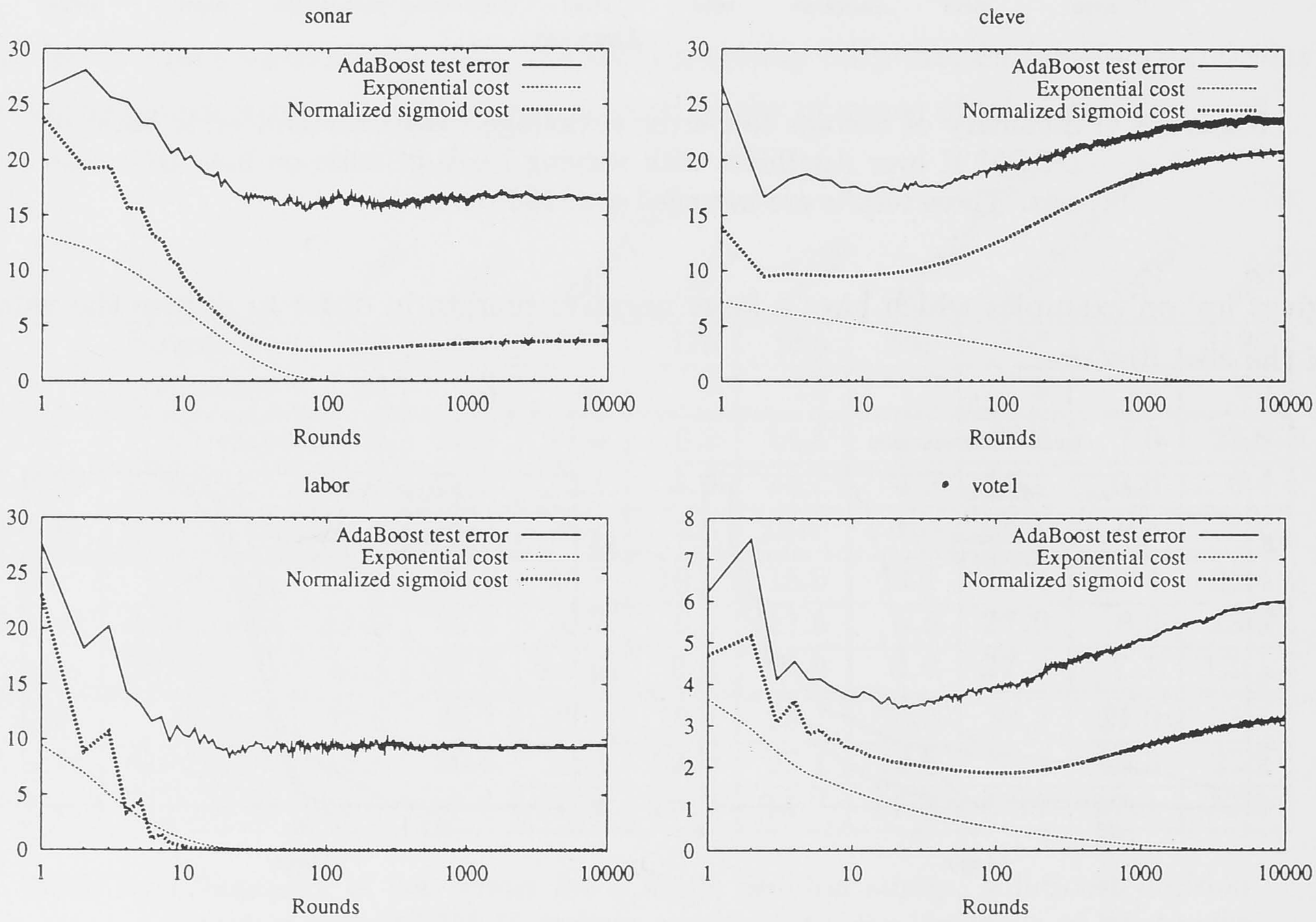


Figure 8.6: AdaBoost test error, exponential cost and normalized sigmoid cost over 10000 rounds of AdaBoost for the `sonar`, `cleve`, `labor` and `vote1` data sets. Both costs have been scaled in each case for easier comparison with test error.

To verify that the choice of step size has little effect on the generalization performance of these gradient descent procedures we compare AdaBoost to a modified version using fixed step sizes, called ε-AdaBoost. In ε-AdaBoost, the first classifier is given weight 1 and all others thereafter are given weight ε. A comparison of the test errors of both of these algorithms for various values of ε is shown in Figure 8.7. As expected, changing the value of the fixed step size ε simply translates the test error curve on the log scale and does not significantly alter the minimum test error.

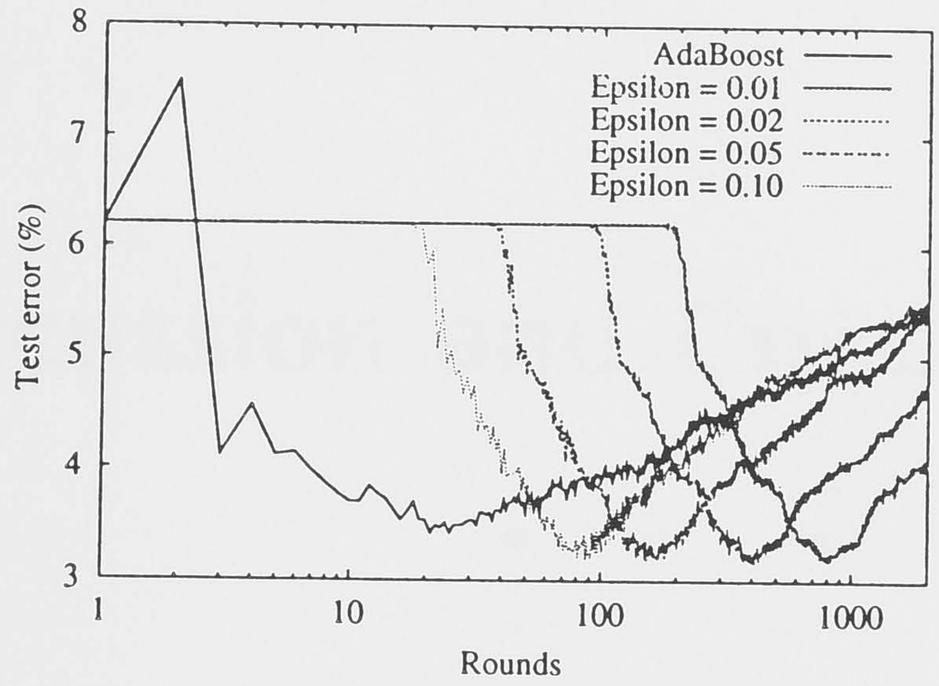


Figure 8.7: Test error for the `vote1` data set over 2000 rounds of AdaBoost and ε-AdaBoost for $\varepsilon = 0.01, 0.02, 0.05$ and 0.10.

8.3 Discussion

Many authors have recently described other techniques for directly optimizing margins (and related quantities). In [57], Rätsch, Onoda and Müller show that versions of AdaBoost modified to use regularization are more robust for noisy data. Friedman, Hastie and Tibshirani [37] and Friedman [36] describe general additive models for regression and classification using various cost functions and present specific cases for voted decision trees. Duffy and Helmbold [26] have developed two algorithms (GeoLev and GeoArc) which attempt to produce combined classifiers with *uniformly* large margins on the training data. In [32], Freund describes a boosting algorithm which uses example weights similar to those suggested by the main theoretical result of Chapter 5. Rätsch *et al.* [58] have developed two algorithms (RoBoost and ν-ARC, based on AdaBoost and Breiman's ARC-GV [15] respectively) which attempt to optimize a margin cost function on all but a fixed proportion of the training data.

CHAPTER

9

Discussion and Conclusions

The work in this thesis can be divided into two main areas.

Firstly, we have derived new upper bounds on the generalization error of several popular classes of classifiers which can be represented as convex combinations, including general voted combinations of classifiers, decision trees and voted combinations of decision trees. These bounds are qualitatively different from those that follow from existing VC theory and depend on the distribution of the margins of the training examples. These bounds can be considerably smaller than existing results when most training examples are classified with a large margin. This shows that existing structural complexity measures for classifiers (such as the number of nodes in a decision tree or the number of classifiers in a voted combination) are not always the most appropriate in determining their generalization error, and that measures of complexity that depend on the training data can sometimes provide a more accurate measure of classifier performance. These bounds may further help to explain the qualitative performance of classifiers which can be represented as convex combinations.

Secondly, we have presented a new understanding of voting methods and developed new algorithms which outperform existing techniques for producing voted combinations of classifiers. By viewing voting methods within an abstract framework for performing gradient descent on cost functionals within an inner product space, we have seen that many existing voting methods are gradient descent optimizers of margin cost functions. In particular, the popular AdaBoost algorithm can be viewed as a procedure for producing voted classifiers which minimize the sample average of an exponential cost function of the training margins. Although AdaBoost is remarkably successful in practice, it can suffer from overfitting, particularly in high noise situations. AdaBoost's exponential cost function often places too much emphasis on examples with large negative margins. The new algorithms we have presented attempt to optimize more sensible cost functions motivated by our new bounds on

the generalization error of convex combinations of classifiers. Experimental results have demonstrated that these algorithms generally outperform AdaBoost, particularly in the presence of label noise.

9.1 Further work

This work leaves several interesting questions unanswered.

Bounds on the generalization error of combined classifiers

Although the theoretical bounds derived help to explain the empirical performance of voting methods, the actual performance is still much better than these bounds would suggest. Proving tighter bounds is an important open problem.

It seems likely that the techniques used in Chapter 5 to derive upper bounds for convex combinations of classifiers in terms of general cost functions of the margin could be applied to specific classes of base classifiers. In particular, the results of Chapters 2 and 3 for decision trees and combinations of decision trees could perhaps be simplified and generalized in terms of more general cost functions of the margin.

Obtaining lower bounds on the generalization error of convex combinations of classifiers in terms of general cost functions of the margin would provide further evidence that these bounds are capturing the important factors which govern generalization error.

Multi-class classification and regression

The theoretical results discussed in Chapters 2, 3 and 5 deal only with binary classification. Extensions of these results to the multi-class case can be obtained using techniques similar to those described in [60]. It seems likely that these results can be further generalized to handle general cost functions on the *distribution* (other than the generalization error) as well as general cost functions on the training set.

The general AnyBoost algorithm described in Chapter 7 makes no restrictions on the label set Y and as such can be applied to multi-class classification and even regression. Suitable choices for cost functions in these more general settings is an open problem. For interesting work in this area see [36, 37].

Decision trees

A better understanding of the complexity of decision trees is still required, the ultimate goal being development of new algorithms which take advantage of these new measures

of complexity. Further analysis of the distribution of training examples over leaves in practical experiments may provide new insights. Theoretically, the analysis seems weaker than it could be. The representation of decision trees as thresholded convex combinations of leaf functions is convenient, but somewhat artificial.

An alternative approach to understanding decision trees in terms of boosting is presented in [23, 42].

Alternating decision trees

There is much work to be done before alternating decision trees can be considered a practical alternative to existing methods. Firstly, the method of construction is currently very computationally intensive. Friedman *et al.* [37] describe a technique called *weight trimming* for reducing the size of the training set used at each iteration of the AdaBoost algorithm. This technique, combined with methods for using sub-optimal splits and cached splits could easily be applied to the algorithm to give significant speed improvements. Secondly, the generalization performance of alternating decision trees is not as good as might be expected. It seems that the voting method used may not be taking full advantage of the power of the class of alternating decision trees. Applying the MarginBoost algorithm with alternative cost functions to the building of alternating decision trees may give improved results.

AnyBoost

The optimization of the nonconvex margin cost functions suggested by the theoretical analysis of Chapter 5 has proven to be difficult. Whether such effort is worthwhile is unclear. An extensive experimental comparison of MarginBoost with various cost functions (both convex and nonconvex) and other regularized voting methods [32, 57, 58] is needed to answer this question.

The AnyBoost.L_2 algorithm is only briefly discussed in this work. Obtaining theoretical results for the case where the combination of classifiers has an L_2 rather than L_1 constraint would be worthwhile given that the L_2 constraint is more natural in an inner product space setting. One important practical advantage of an L_2 constraint is that it allows for base learners to return general linear combinations of base classifiers at each iteration. In Chapter 7 we described the InnerBoost algorithm, which converts a base learner returning a single base classifier to a base learner returning linear combinations of base classifiers. Further experimental analysis is needed to evaluate whether this gives any significant improvement in convergence time.

The generality of the AnyBoost algorithm means that it can be applied to any inner product space, not just spaces of functions. This opens up the possibility of applying

boosting techniques to many other machine learning problems.

Complexity of the base classifiers in voted combinations

All of the voting method experiments presented in this thesis used decision stumps as base classifiers. Recent results [15] have shown that using more complex base classifiers (such as decision trees) can significantly affect the relative performance of voting methods. Understanding the possible interactions between a voting method and its class of base classifiers is an important open problem.

Bibliography

[1] D. Aldous and U. Vazirani. A Markovian extension of Valiant's learning model. In *Proceedings of the 31st Symposium on the Foundations of Computer Science*, pages 392–396. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[2] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 1999.

[3] M. Anthony and J. Shawe-Taylor. A result of Vapnik with applications. *Discrete Applied Mathematics*, 47:207–217, 1993.

[4] P. L. Bartlett. Learning with a slowly changing distribution. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 243–252. ACM Press, 1992.

[5] P. L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[6] P. L. Bartlett, S. Ben-David, and S. R. Kulkarni. Learning changing concepts by exploiting the structure of change. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 131–139. ACM Press, 1996.

[7] P. L. Bartlett, P. Fischer, and K. Höffgen. Exploiting random walks for learning. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory*, pages 318–327. ACM Press, 1994.

[8] R. D. Barve and P. M. Long. On the complexity of learning from drifting distributions. *Information and Computation*, 138(2):101–123, 1997.

[9] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1):105–139, 1997.

[10] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation 1*, pages 150–160, 1989.

[11] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[12] A. Blum and P. Chalasani. Learning switching concepts. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 231–242. ACM Press, 1992.

[13] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[14] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[15] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7), 1999. (to appear).

[16] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[17] W. .L Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[18] W. L. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992.

[19] M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin-Madison, 1996. Also appears as UW Technical Report CS-TR-96-1326.

[20] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Applications of Mathematics: Stochastic Modelling and Applied Probability (31). Springer, New York, 1996.

[21] L. Devroye and G. Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition*, 28, 1995.

[22] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 1999. (to appear).

[23] T. G. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proceedings of the 13th International Conference on Machine Learning*, pages 96–104. Morgan Kaufmann, 1996.

[24] P. Domingos. Knowledge acquisition from examples via multiple models. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 98–106, 1997.

[25] H. Drucker and C. Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, pages 479–485. Morgan Kaufmann, 1996.

[26] N. Duffy and D. Helmbold. A geometric approach to leveraging weak learners. In *Computational Learning Theory: 4th European Conference*, 1999. (to appear).

[27] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989.

[28] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

[29] U. M. Fayyad and K. B. Irani. What should be minimized in a decision tree? In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 249–754, 1990.

[30] U. M. Fayyad and K. B. Irani. The attribute selection problem in decision tree generation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 104–110, 1992.

[31] M. Frean and T. Downs. A simple cost function for boosting. Technical report, Department of Computer Science and Electrical Engineering, University of Queensland, 1998.

[32] Y. Freund. An adaptive version of the boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 1999. (to appear).

[33] Y. Freund and Y. Mansour. Learning under persistent drift. In *Computational Learning Theory: 2nd European Conference*, pages 109–118. Springer-Verlag, 1997.

[34] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[35] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[36] J. Friedman. Greedy function approximation: A gradient boosting machine. Technical report, Stanford University, 1999.

[37] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. Technical report, Stanford University, 1998.

[38] A. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

[39] D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.

[40] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[41] M. Kearns. *The Computational Complexity of Machine Learning*. MIT Press, 1990.

[42] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1996.

[43] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41(1):67–95, January 1994.

[44] R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 161–169, 1997.

[45] A. Kowalczyk and H. Ferrá. Developing higher order networks with empirically selected units. *IEEE Transactions on Neural Networks*, 5(5):698–711, September 1993.

[46] W. S. Lee, P. L. Bartlett, and R. C. Williamson. Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Transactions on Information Theory*, 42(6):2118–2132, November 1996.

[47] R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551, 1997.

[48] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 211–218, 1997.

[49] L. Mason, P. L. Bartlett, and M. Golea. Generalization in threshold networks, combined decision trees and combined mask perceptrons. In *Proceedings of the Ninth Australian Conference on Neural Networks*, pages 84–88, 1998.

[50] L. Mason, P. L. Bartlett, and M. Golea. Generalization error of combined classifiers. *Journal of Computer and System Sciences*, 1999. (to appear).

[51] J. Mingers. An empirical comparison of selection measures for decision tree induction. *Machine Learning*, 3(4):319–342, 1989.

[52] P. M. Murphy and M. J. Pazzani. Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction. *Journal of Artificial Intelligence Research*, 1:257–275, 1994.

[53] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[54] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[55] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[56] G. Rätsch. Ensemble learning methods for classification. Master's thesis, Department of Computer Science, University of Potsdam, April 1998. (in German).

[57] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. Technical Report NC-TR-1998-021, Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998.

[58] G. Rätsch, B. Schökopf, A. Smola, S. Mika, T. Onoda, and K.-R. Müller. Ro-Boost and ν-Arc: Ensemble learning in the presence of outliers. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 1999. (to appear).

[59] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory Series A*, 13:145–147, 1972.

[60] R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, October 1998.

[61] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998.

[62] H. Schwenk and Y. Bengio. Training methods for adaptive boosting of neural networks. In *Advances in Neural Information Processing Systems 10*, pages 647–653. MIT Press, 1998.

[63] J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony. A framework for structural risk minimisation. In *Proc. 9th COLT*, pages 68–76. ACM Press, New York, NY, 1996.

[64] S. Shelah. A combinatorial problem: Stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41:247–261, 1972.

[65] H. U. Simon. General bounds on the number of examples needed for learning probabilistic concepts. *Journal of Computer and System Sciences*, 52(2):239–254, 1996.

[66] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[67] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

[68] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[69] V. N. Vapnik. *Statistical Learning Theory.* John Wiley, 1998.

[70] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[71] V. N. Vapnik and A. Y. Chervonenkis. *Theory of Pattern Recognition.* (in Russian) Nauka, Moscow, 1974. German Translation: *Theorie der Zeichenerkennung*, Akadamie Verlag, Berlin, 1979.

[72] M. Vidyasagar. *A Theory of Learning and Generalization, with Applications to Neural Networks and Control Systems.* Springer, New York, 1997.

[73] G. L. Webb. Further experimental evidence against the utility of Occam's razor. *Journal of Artificial Intelligence Research*, 4:397–417, 1996.